Experimental Implementation of Formation Control Algorithm on UAV Testbed

A Thesis

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree

Master of Science in Electrical Engineering

by

Boakye Dankwa

UNIVERSITY OF DAYTON

Dayton, Ohio

May, 2008

Experimental Implementation of Formation Control Algorithm on UAV Testbed

APPROVED BY:

Raúl Ordóñez, Ph.D.
Advisory Committee Chairman
Associate Professor
Department of Electrical and
Computer Engineering

John S. Loomis, Ph.D.
Committee Member
Associate Professor
Department of Electrical and
Computer Engineering

Russell C. Hardie, Ph.D
Committee Member
Professor
Department of Electrical and Computer Engineering

Malcolm W. Daniels, Ph.D.
Associate Dean
Graduate Engineering Program &
Research, School of Engineering

Joseph E. Saliba, Ph.D., P.E.
Dean, School of Engineering

To my parents.

ABSTRACT

Although there has been a lot of research in multi-agent coordination in recent years, practical implementation of proposed algorithms is rare in the literature. This thesis presents an experimental implementation of the control strategy for a multi-agent system to capture a moving target in specific formation as presented in [1]. The problem considered is that of implementing a decentralized control algorithm on a Uninhabited Air Vehicle (UAV) testbed at the Wright Patterson Air-force Base (WPAFB). Point mass dynamics are used to model the planner motion of the UAVs. An attraction-repulsion potential profile is used to maintain the formation and orientation and also prevent collisions between the agents. A virtual target is used to generate the desired trajectory for the agents to follow in formation. The control algorithm generates the formation trajectories which are sent over a wireless communication network to the UAVs on-board autopilot in real-time for navigation. Comparisons of the experimental results and simulation verify the robustness of the algorithm.

ACKNOWLEDGMENTS

My foremost thank goes to my advisor Professor Raúl Ordóñez. I thank him for his patience and encouragement that carried me on through the masters program, and for his insights and suggestions that helped to shape my research skills. Without his support, this dissertation would not have been possible. I appreciate his passion in control, academic attitude and pleasant personality. I am really glad that I got to know professor Ordóñez in my life.

This research at University of Dayton is made possible by the financial support from Wright Paterson Air Force Base (WPAFB). Special thanks to Lt. Scott Pilukaitis and Lt. Majo Sean R, for their administrative and technical support at the WPAFB, especially in setting up the UAV network for the ground tests.

I express my deepest gratitude to you Professor John Loomis and Professor Russell Hardie at University of Dayton, for serving in my thesis committee and helping me improve this thesis. Thank you Prefessor Loomis for your introduction to the Qt3 software.

I am grateful to former PhD student Shreecharan Kanchanavally, who introduced and helped install Player/Gazebo simulation software. Special thanks to you Marilyn K. Knisley and Loretta P. Christon, for their support in administrative and human resource services.

Finally I would like to thank you all my friends and colleagues in Control group, Jingyi Yao, Kayode Ajayi-Majebi, Emmanuel Otoo and Sudarshan Srinivasan for the wonderful association and fruitful technical discussions that helped me in various aspects of this research.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Consider the following scenario: *A city has been hit by a natural disaster. The conditions there are extremely dangerous for humans to mount a search and rescue operation. Fortunately we have perfected the technology for advanced robotics and an "army" of robots capable of performing any task is available. These robots are capable of autonomous operation and can perform complicated tasks that none of them can do alone. A team of robots are sent to the city to start searching for survivors and conduct preliminary preparation before human workers can be sent. Survivors are rescued and the city is cleared enough for humans to take over the operations. The objective has been accomplished.* The scenario described above involves many technologies including robotics, decision and control, artificial intelligence, communication, sensing, to mention a few. One of the critical areas is cooperative control.

Cooperative control is the coordination of multiple agents to achieve a common goal. Research in this area has gained lots of attention in the control community in recent years. Many complex engineering problems could be solved more efficiently and cheaply using cooperative autonomous agents as opposed to using a single, large and

expensive one. Much of the research in cooperative control draws motivation from biological systems. The complexities of biological systems are non-trivial and designing engineering systems to exhibit behavior of such systems is challenging. Challenging as it is, many important results have been derived from studying organisms in the environment in recent years in an attempt to understand their behavior and apply the knowledge gained to solve complex engineering problems. For instance, studies show that flocking and schooling are beneficial to the animals that use them in that they make searching for food and defense against predators more efficient. It is amazing to imagine how a swarm of geese fly in formation closely together without colliding with each other. They must have some form of communication and coordination to direct and maintain the formation in flight. Artificial agents can be designed to take advantage of these behaviors to accomplish useful tasks.

Applications of the control and coordination of cooperative autonomous vehicles in our present day cannot be over-emphasized. There is intensive ongoing research in many military as well as civilian institutions on applications of control of multiple autonomous vehicles. Possible applications include coordinated control of UAVs for purposes like surveillance, distribution of sensor networks, automated highways, satellite formation, coordinated movement of a cluster of vehicles for the search and rescue operation and space exploration using a team of autonomous vehicles. Though the technology is not perfected yet, it is important to actually implement some of these proposed algorithms on experimental robots. So far practical implementation of research in this area is rare in the literature. The work in this thesis addresses the problem of implementing a formation control algorithm, which is a special case of cooperative control, on a UAV testbed at the WPAFB.

1.2 Literature Overview

Multi-agent coordination has received a lot of attention recently mostly due to its critical importance to military and civilian applications. A lot of research work is ongoing in this area to solve the problem of coordination and control of Uninhabited Autonomous Vehicles (UAVs). Formation control, one of the popular branches of multi-agent coordiantion, is the main object of this thesis. The research reported in this thesis benefits from the work of Jingyi Yao et al [1]. In this work, the authors proposed a formation control scheme using potential functions for a group of autonumous agents to capture a moving target and maintain a specified formation around the target. Here the authors first developed a control scheme for agents with a kinematic model and proved convergence using LaSalle's theorem. The sliding mode technique was later used to drive the dynamics of realistic agents to track the states of the kinematic model. They provided conditions to guarantee that the target will always remain in the convex hull of the formation. The potential function used here was symetric and a function of relative distance. This means the orientation of the formation is arbitrary. In [2], the authors used virtual leaders to enforce formation orientation. They defined an additional potential which depends on the angle between a reference line attached to the kth virtual leader and the vector defining the distance between that virtual leader and the ith agent. In this thesis, we extend the work in [1] to control formation orientation by defining reference points reletive to the target position. This thesis implements the extended scheme on a UAV testbed at the WPAFB.

Using potential fields (Navigation Function) for motion planning was first pioneered by Khatib and LeMiatre [3]. Khatib treated the robot as a point in configuration space under the influence of a potential field whose profile represents the 'structure' of the free space. The potential field is typically the sum of attractive potential pulling the robot towards the goal configuration and a repulsive potential pushing the robot away from obstacles. The potential function approach is basically solving a gradient descent problem where the motion of the agents is along the negative gradient of a potential function. Recent studies have extended potential field methods to the maneuvering of group behaviors such as formation, migration and obstacle avoidance in swarm systems [4, 5, 6, 7]. Formation information can be encoded in the potential function, thus the required formation follows when the potential reaches its minimum configuration. Such an approach is presented as a direct application of the work done by V. Gazi [6]. Here the author discusses swarm aggregation where a group of agents each represented by a first order kinematic equation moves along the negative gradient of a potential function. The potential function is based on the relative distances of the agents from each other and must be symmetric and consist of attraction and repulsion components. The author stated that the center of the swarm is stationary for all time and that the agents will come to a halt in finite time for any initial conditions and also if the potential function is chosen such that it has a unique minimum at a desired swarm configuration, then such a formation will be achieved asymptotically. The case of agents with realistic dynamics was considered where the sliding mode control technique was used to force the motion of the agent to track that of the ideal kinematic agent. His later work [8] presented stability analysis of swarm in Lyapunov's perspective and a bound on the swarm size was obtained.

The major problems acknowledged by almost every robot navigation paper using the potential function approach is the local minima/maxima problem [9].

Most of the potential functions suitable for robot navigation have multiple minima, this is undesired because the system's state can get locked at such an undesired minimum, resulting in an undesired behavior. A clever way of solving this problem is presented by Rimon and Koditschek in [10]. The authors used advanced mathematics to construct differentiable navigation functions which are free of local minima except the global minimum. It attains its maximum at all obstacle boundaries. Rimon and Koditschek used Lyapunov theory to prove convergence. Bounded control and safety was also shown. The authors proved that once such a navigation function is constructed, the path planning problem is solved and obstacle avoidance will be guaranteed. The construction of navigation function however, is based on 'generalized sphere worlds' containing obstacles of specific categories. Adapting the scheme to robots with realistic dynamics and obstacles with arbitrary dimensions presents a formidable challenge. In [11], the authors tackled the navigation problem by considering an equivalent electrostatic problem and establishing a navigation function which is devoid of local minima and analogous to a scalar electrical potential over a domain free of electric static charges and constituded by a single dielectric isotropic material. The resulting problem is then solved with the finite element method.

Graph theory on the other hand, has been a powerful tool for researchers in formation control. In [12, 13] Kumar et al proposed a formation control scheme using feedback laws and graph theory in a leader-follower fashion on nonholonomic vehicles to navigate a terrain with obstacles. Olfati-Saber and Murray [14] also developed a

formation control stratagy using graph theory. Formation stability was proved using Lyapunov theory.

The sliding mode control technique is a robust control technique which has a special property of insensitivity to model uncertainties and disturbances. In [15, 16], it was used for robot navigation and obstacle avoidance in an environment modeled with harmonic potentials.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: In Chapter 2, we develop the procedure for controlling formation orientation and show that stability results previously obtained in [1] are preserved. A review of the previously proposed controller is given for completeness and Matlab simulations results are shown. In Chapter 3, we show software development of the formation control algorithm. The structure of the code is discussed and the functions of a custom Graphical User Interface (GUI) developed for the formation control code is also given. In Chapter 4, we implement the formation control scheme in a virtual environment and show simulation results. In Chapter 5, we implement the scheme on the UAV test bed and results obtained from experiments are compared with those from simulations. In Chapter 6, conclusions are made and recommendations for future research are also discussed.

1.4 Contribution

The work in this thesis can be considered as an extension to the work in [1]. We develop a technique for controlling formation orientation and implemented the algorithm on a UAV test bed. Direction finding may be a direct application of this

research. Controlling the distance and orientation between two helicopters can effectively manage the baseline of an antenna system and create a much larger antenna array with multiple helicopters than will be posible with a single helicopter. The custom GUI developed for this research may be further developed to serve as an independent simulation tool for coorperative control algorithms.

1.5 Previous Work

The initial research at the WPAFB was to develop software to compute the shorterst distance between a flying UAV and a reference position in real-time using GPS position data from the UAV. This was a precursor to the main research and served as an introduction to the Linux Operating System and the experimental helicopter API software environment. This section discusses the work that was done.

1.5.1 Position Reference Program

Let the domain $D \in \mathbb{R}^3$ be the Local Tangent Plane and x_{pos} and x_{ref} be the position of the UAV and a reference position respectively in D. By definition, the Euclidean distance between x_{pos} and x_{ref} is

$$r = \|x_{pos} - x_{ref}\|. \tag{1.1}$$

A simple flowchat that implements Equation (1.1) can be found in Appendix-I. The filtered GPS data arrived at a rate of 4Hz and the origin of the Local Tangent Plane was usually the reference position. A ground test of the program was conducted on Area B Runway at the WPAFB. Cones were used to mark known distances from a reference point on the runway. The UAV was then moved to each of the cones while the program was running to check the output of the program against the known

distances. Figure (1.1) shows a plot of data collected during the ground test. The C++ code and raw data collected can also be found in Appendix-I.

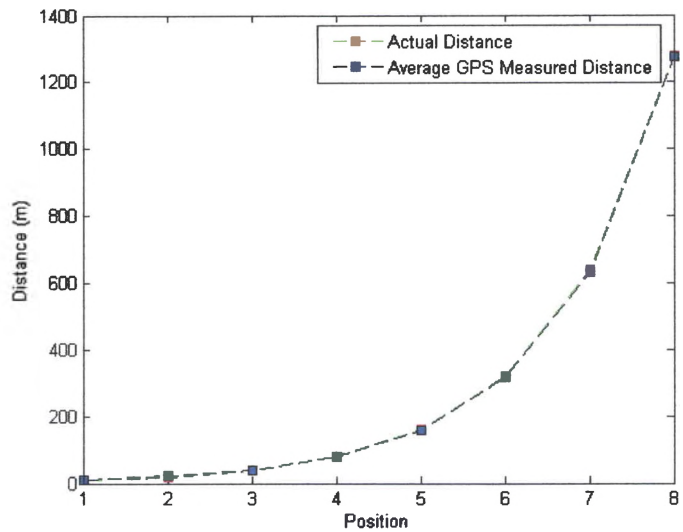


Figure 1.1: Results from position reference code showing average GPS measured distance and actual distance.

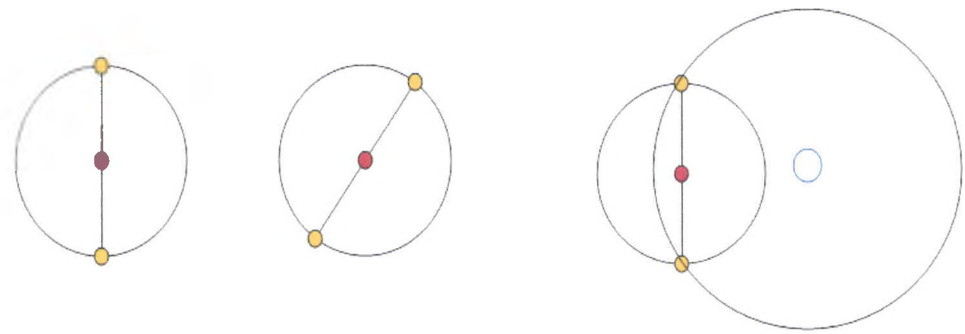
CHAPTER 2

COORDINATION STRATAGY

The coordination stratagy adapted for this research was first proposed by J. Yao et al [1] at the Non-linear Systems and Control Lab at the University of Dayton. In [1], the authors proposed a stable decentralized stratagy for multi-agent system to capture a moving target in a specific formation. Artificial potential functions were used to enforce both tracking and formation tasks. They first establish the control stratagy for a basic "kinematic model" and later use the sliding mode technique to drive the dynamics of a class of agents vehicle dynamics to track the dynamics of the kinematic model. This chapter discusses the extensions made to the "kinematic model" and shows that stability results obtained in [1] were perserved. Matlab simulation examples are provided to illustrate the technique adapted.

2.1 Formation Orientation

The formation stratagy proposed by Yao et al depends on the relative distances between agents and between agents and target. As shown in Figure 2.1(a), the equilibrium solutions for such formation has $\mathbb{S}^1$ symmetry and a family of solutions [2]. To break this symmetry and fix the orientation of the agents, additional reference positions need to be defined as shown in Figure 2.1(b). Let the position of the ith



(a) (b)

Figure 2.1: Equilibrium solutions for a formation in 2D with two vehicles. (a) With no reference point there is S^1 symmetry and a family of solutions. (b) With at least one reference point S^1 symmetry can be broken and orientation of the group fixed.

vehicle in a group of N vehicles, with respect to an inertial frame be given by a vector $x_i \in \mathbb{R}^n, i = 1, ..., N$ as shown in Figure 2.2. Assuming the point mass model, the control force on the ith vehicle is given by $u_i \in \mathbb{R}^n$ and the motion dynamics can be written as

$$\dot{x}_i = u_i.$$

The position of the target is $x_t \in \mathbb{R}^n$, and let the position of the kth reference point be $b_k \in \mathbb{R}^n, k = 1, ..., N$ all with respect to the inertial frame. In [2] the authors defined the orientation of the vehicles as dependent on the orientation of the virtual leaders with respect to the virtual body frame oriented as the inertial frame but with origin at the center of mass of the virtual body. Here, the frame of the reference points, refered to as "reference frame" in this thesis, is oriented as the inertial frame, however, it is centered at the position of the target. The orientation of the plane of formation about the target can be changed by changing the orientation of the "reference frame" about

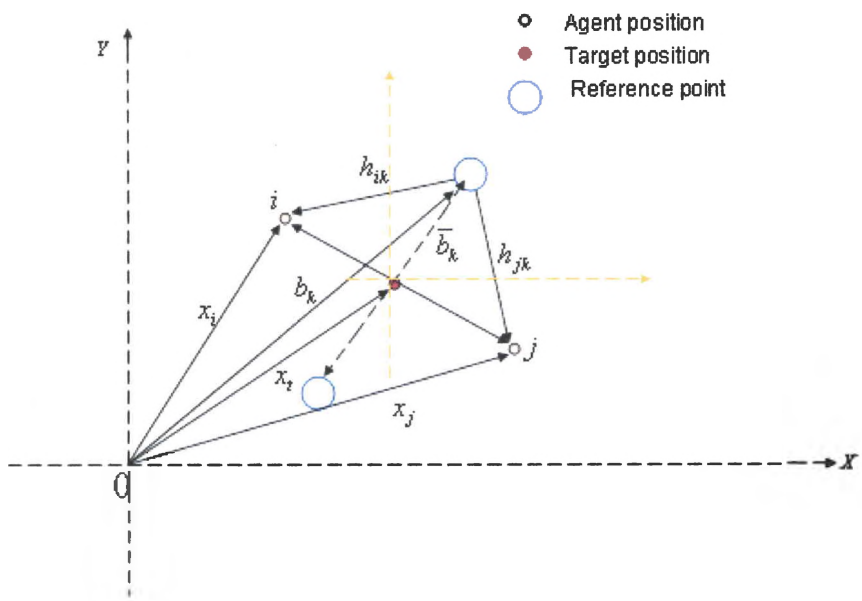


Figure 2.2: Notation for framework

the target. Let the position of the kth reference position be defined in the "reference frame" by the constant[1] vector $\bar{b}_k \in \mathbb{R}^n$ Thus the position of the kth reference point in the inertial frame is given by

$$b_k = x_t + \bar{b}_k. \tag{2.1}$$

The potential, $J_{ik}(\|x_i - b_k\|)$ was introduced in this work for orientation control to augument the potential proposed in [1]. Assume $J_{ik}(\|x_i - b_k\|)$ has the following properties:

- There exist corresponding function $f^{ik} : \mathbb{R}^+ \to \mathbb{R}$ such that

$$\nabla_y J_{ik}(\|y\|) = y f^{ik}(\|y\|). \tag{2.2}$$

[1] $\bar{b}_k$ is constant here so we can preserve the stability results obtained in [1]. The case where $\bar{b}_k$ is time dependent is a topic for future research

- There exist unique distances $y = h_{ik}$ at which we have $f^{ik}(\|y\|) = 0$.

Definition (Potential function): $J : \mathbb{R}^{n \times N} \times \mathbb{R}^n \to \mathbb{R}$ *is a differentiable, nonnegative, decrescent, radially unbounded function of the distance* $\|x_i - x_t\|$ *between agent* i *and target, the distance* $\|x_i - x_j\|$ *between agent* i *and* j *and the distance* $\|x_i - b_k\|$ *between agent* i *and the* k*th reference point such that*

- J *has only one minimum and attains its unique minimum when* $\|x_i - x_t\| = \delta_{it}$, $\|x_i - x_j\| = \delta_{ij}$ *and* $\|x_i - b_k\| = h_{ik}$ *for all* i, j,*and* k $(i, j, k = 1, \cdots, N)$.

- $\nabla_{x_t} J(x, x_t) = -\sum_{i=1}^{N} \nabla_{x_i} J(x, x_t)$.[2]

$J(x, x_t)$ is of the form

$$J(x, x_t) = K_T \sum_{i=1}^{N} J_{it}(\|x_i - x_t\|) + K_F \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} J_{ij}(\|x_i - x_j\|) + K_V \sum_{i=1}^{N} \sum_{k=1}^{N} J_{ik}(\|x_i - b_k\|) \tag{2.3}$$

$J_{it}(\|x_i - x_t\|)$ is the potential between agent and target, $J_{ij}(\|x_i - x_j\|)$ is the potential between agents and $J_{ik}(\|x_i - b_k\|)$ is the potential between the agents and the reference points. The coefficients K_T, K_F and K_V weigh the relative importance of tracking, formation and orientation. The overall state x is implicitly defined as $x^\top = [x_1^\top, \ldots, x_N^\top] \in \mathbb{R}^{n \times N}$. When we are able to drive $J(x, x_t)$ to the global minimum, each agent reaches the desired distances from both the target and the other agents. Though the local minimum problem is always a concern in using potential functions to coordinate autonomous robots, it may be reduced by careful choice of the potential function and the initial conditions of the agents. Note that from Figure 2.2, $\underbrace{[h_{ik}]}_{1 \le i,k \le N} = H$ is a matrix and we will refere to it as the "reference matrix" in the remender of this thesis.

[2]This will be derived for J in the next section.

2.2 Controller Design

Following the definitions of $J_{ij}(\|x_i - x_j\|)$ and $J_{it}(\|x_i - x_t\|)$ in [1] and taking the partial derivative of $J(x, x_t)$, we have

$$\begin{aligned}
\nabla_{x_i} J(x, x_t) &= K_T(x_i - x_t)h^{it}(\|x_i - x_t\|) \\
&+ K_F \sum_{j=1, j\neq i}^{N} (x_i - x_j)g^{ij}(\|x_i - x_j\|) \\
&+ K_V \sum_{k=1}^{N} (x_i - b_k)f^{ik}(\|x_i - b_k\|) \qquad (2.4) \\
\nabla_{x_t} J(x, x_t) &= -K_T \sum_{i=1}^{N} (x_i - x_t)h^{it}(\|x_i - x_t\|) \\
&-K_V \sum_{i=1}^{N}\sum_{k=1}^{N} (x_i - b_k)f^{ik}(\|x_i - b_k\|). \qquad (2.5)
\end{aligned}$$

Equation (2.5) follows from equation (2.1). By observing the quantities in (2.4) and (2.5), we see that

$$\begin{aligned}
\nabla_{x_t} J(x, x_t) &= -\sum_{i=1}^{N} \nabla_{x_i} J(x, x_t) \\
&+K_F \sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N} (x_i - x_j)g^{ij}(\|x_i - x_j\|). \qquad (2.6)
\end{aligned}$$

Moreover, since we have [1]

$$\sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N} (x_i - x_j)g^{ij}(\|x_i - x_j\|) = 0, \qquad (2.7)$$

we then obtain the useful relationship:

$$\nabla_{x_t} J(x, x_t) = -\sum_{i=1}^{N} \nabla_{x_i} J(x, x_t). \qquad (2.8)$$

Taking the time derivative of J and substituting the condition (2.8) in the $\dot{J}$ equation, one obtains

$$\begin{aligned}\dot{J} &= \sum_{i=1}^{N}[\nabla_{x_i}J(x,x_t)]^\top \dot{x}_i + [\nabla_{x_t}J(x,x_t)]^\top \dot{x}_t \\ &= \sum_{i=1}^{N}[\nabla_{x_i}J(x,x_t)]^\top u_i - \sum_{i=1}^{N}[\nabla_{x_i}J(x,x_t)]^\top \dot{x}_t \\ &= \sum_{i=1}^{N}[\nabla_{x_i}J(x,x_t)]^\top (u_i - \dot{x}_t). \end{aligned} \tag{2.9}$$

Since $\dot{x}_t$ is unknown in most cases it is more realistic to assume that $\|\dot{x}_t\| \leq \gamma_t$ for some known $\gamma_t > 0$. With this assumption we can choose the control law as

$$u_i = -\alpha\nabla_{x_i}J(x,x_t) - \beta\text{sign}(\nabla_{x_i}J(x,x_t)), \tag{2.10}$$

for all $i = 1,...,N$ where $\alpha > 0$ and $\beta \geq \gamma_t$ are positive constants, sign$(\cdot)$ is the signum function operated elementwise for a vector $y \in \mathbb{R}^n$, i.e., $sign(y) = [sign(y_1),...,sign(y_n)]^\top$. It can be shown using *Lasalle-Yoshizawa* theorem [1] that as $t \to \infty$ we have $(x,x_t) \to \Omega \subset \{(x,x_t)|\dot{J} = 0\}$ where

$$\Omega = \{(x,x_t)|\nabla_{x_t}J(x,x_t) = 0, \nabla_{x_i}J(x,x_t) = 0, i = 1...N\}.$$

Since we assume the potential function has only one minimum (in relative distance coordinates), it indicates J converges to the unique global minimum, implying that the system converges to a configuration corresponding to that minimum.

2.3 Potential Function Discussions

2.3.1 Eliminating the $J_{ij}(\|x_i - x_j\|)$ Terms

It is important to note that the term $J_{ik}(\|x_i - b_k\|)$ addresses both formation and orientation of the agents. This is because fixing the positions of the agents with

respect to the reference points result in the agents being in a specific formation and orientation around the target. We can redefine $J(x, x_t)$ without the J_{ij} term as

$$J(x, x_t) = K_T \sum_{i=1}^{N} J_{it}(\|x_i - x_t\|) + K_V \sum_{i=1}^{N} \sum_{k=1}^{N} J_{ik}(\|x_i - b_k\|). \tag{2.11}$$

Taking the partial derivative of $J(x, x_t)$, we have

$$\begin{aligned} \nabla_{x_i} J(x, x_t) &= K_T(x_i - x_t)h^{it}(\|x_i - x_t\|) \\ &+ K_V \sum_{k=1}^{N} (x_i - b_k) f^{ik}(\|x_i - b_k\|) \end{aligned} \tag{2.12}$$

$$\begin{aligned} \nabla_{x_t} J(x, x_t) &= -K_T \sum_{i=1}^{N} (x_i - x_t)h^{it}(\|x_i - x_t\|) \\ &-K_V \sum_{i=1}^{N} \sum_{k=1}^{N} (x_i - b_k) f^{ik}(\|x_i - b_k\|). \end{aligned} \tag{2.13}$$

We see that

$$\nabla_{x_t} J(x, x_t) = -\sum_{i=1}^{N} \nabla_{x_i} J(x, x_t),$$

which was obtained in Equation (2.8). This means we could eliminate the $J_{ij}(\|x_i - x_j\|)$ terms from $J(x, x_t)$ and still achieve formation and orientation. Though this will result in a less complicated total potential function, which leads to less computation time during implementation, it however translates into less formation robustness. The $J_{ij}(\|x_i - x_j\|)$ terms allow us to introduce repulsive forces between the agents which make the formation more robust and also help to reduce the likelihood of inter-agent collission. A potential function of the form given in equation (2.3) was adapted for this work.

2.3.2 Choosing $J(x, x_t)$ for Implementation

For practical purposes, the choice of the total potential function, $J(x, x_t)$, is very important. In the context of this thesis, the choice of the formation potential function

$J_{ij}(\|x_i - x_j\|)$ is crucial. This must be chosen carefuly so that the designer can control the extent of attractive and repulsive forces between the agents. Besides satisfying all the conditions in [1], it should also be bounded from infinity to prevent actuator saturation when the agents get too close to each other. Thus

$$\lim_{y \to 0} J_{ij}(\|y\|) \leq \bar{J}_{ij} < \infty. \tag{2.14}$$

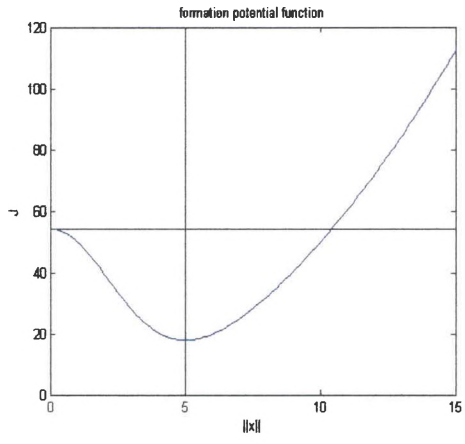
where $\overline{J_{ij}}$ is a positive constant. We choose an attraction-repulsion potential function,

$$J_{ij}(\|x_i - x_j\|) = \frac{a_{ij}}{2}\|x_i - x_j\|^2 + \frac{b_{ij}c_{ij}}{2} exp(-\frac{\|x_i - x_j\|^2}{c_{ij}}) \tag{2.15}$$

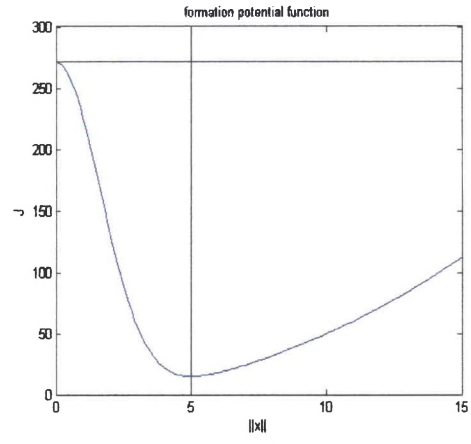
as a candidate for the formation potential. The distance $\delta_{ij} = \sqrt{c_{ij} ln(b_{ij}/a_{ij})}$ is the distance at which the attraction and repulsion balance [17, 18], which also defines the distance between the agents in the desired formation. The relative degree of attraction or repulsion is determined by the choice of the parameters a_{ij}, b_{ij} and c_{ij}, thus for a given formation, we can choose them to have a more profound repulsion as to attraction as shown in the one-dimensional plot of equation (2.15) in Figure 2.3. We can take advantage of this property to prevent inter-agent collision by making the repulsive forces between the agents to be very high as they close in on each other, though within the contraints of equation (2.14). Quadratic potentials were chosen for $J_{it}(\|x_i - x_t\|)$ and $J_{ik}(\|x_i - b_k\|)$ as

$$J_{it}(\|x_i - x_t\|) = \frac{1}{2}(\|x_i - x_t\|^2 - \delta_{it}^2)^2$$

$$J_{ik}(\|x_i - x_t\|) = \frac{1}{2}(\|x_i - b_k\|^2 - h_{ik}^2)^2.$$



(a) $a_{ij} = 1, b_{ij} = 10$ and $c_{ij} = \frac{25}{\ln 10}$



(b) $a_{ij} = 1$, $b_{ij} = 100$ and $c_{ij} = \frac{25}{\ln 100}$

Figure 2.3: One-dimensional plot of formation potential

and the total potential of the system becomes

$$\begin{aligned} J(x, x_t) &= K_T \sum_{i=1}^{N} \frac{1}{2}(\|x_i - x_t\|^2 - \delta_{it}^2)^2 + \\ & K_F \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{a_{ij}}{2}\|x_i - x_j\|^2 + \frac{b_{ij}c_{ij}}{2} exp(-\frac{\|x_i - x_j\|^2}{c_{ij}}) + \\ & K_V \sum_{i=1}^{N} \sum_{k=1}^{N} \frac{1}{2}(\|x_i - b_k\|^2 - h_{ik}^2)^2. \end{aligned} \tag{2.16}$$

2.4 Vehicle with General Dynamics

With the necessary extensions made to the "kinematic model." it was shown that the control force applied to each agent is essentially identical to the control force proposed in [1]. Hence extension to agents with vehicle dynamics remains the same. It is repeated here for completeness. Consider all the agents in the system have the same dynamics, which could be described by the equation

$$M(x_i)\ddot{x}_i + f_i(x_i, \dot{x}_i) = u_i, \tag{2.17}$$

where $x_i \in \mathbb{R}^n$ is the position of the agent, $M(x_i) \in \mathbb{R}^{n\times n}$ is the mass or inertia matrix, and $u_i \in \mathbb{R}^n$ represents the control inputs (forces). The function $f_i(x_i, \dot{x}_i) \in \mathbb{R}^n$ represents centripetal forces, Coriolis, gravitational affects and additive disturbances. For this term we assume that

$$f_i(x_i, \dot{x}_i) = f_i^k(x_i, \dot{x}_i) + f_i^u(x_i, \dot{x}_i)$$

where $f_i^k(\cdot, \cdot)$ represents the known part and $f_i^u(\cdot, \cdot)$ represents the unknown part. For the unknown part, we assume that $\|f_i^u(x_i, \dot{x}_i)\| \leq \overline{f}_i$, where $\overline{f}_i < \infty$ is a known constant. Moreover, it is assumed that for all x_i the matrix $M(x_i)$ satisfies

$$\underline{M}\|y\|^2 \leq y^\top M(x_i) y \leq \overline{M}\|y\|^2,$$

where $\underline{M}$ and $\overline{M}$ are known and $y \in \mathbb{R}^n$ is arbitrary. Note that all these assumptions are standard and realistic. Define the n-dimensional sliding manifold for the ith agent as

$$s_i = \dot{x}_i + \alpha \nabla_{x_i} J(x, x_t) + \beta sign(\nabla_{x_i} J(x, x_t)). \tag{2.18}$$

On the sliding manifold, $s_i = 0$, we have

$$\dot{x}_i = -\alpha \nabla_{x_i} J(x, x_t) - \beta sign(\nabla_{x_i} J(x, x_t)), \tag{2.19}$$

which is the motion equation of the "kinematic model." A sufficient condition to reach the sliding surface asymptotically is given by

$$s_i^\top \dot{s}_i < 0. \tag{2.20}$$

Differentiating equation 2.18 we have,

$$\dot{s}_i = \ddot{x}_i + \frac{\partial}{\partial t}[\alpha \nabla_{x_i} J(x, x_t)] + \frac{\partial}{\partial t}[\beta sign(\nabla_{x_i} J(x, x_t))] \tag{2.21}$$

and rearranging equation 2.17, we obtain

$$\ddot{x}_i = M^{-1}(x_i)[u_i - f_i(x_i, \dot{x}_i)]$$

which when subsutited along with equation 2.21 into equation 2.20, yields

$$s_i^\top \dot{s}_i = s_i^\top [M^{-1}(x_i)[u_i - f_i(x_i, \dot{x}_i)] + \frac{\partial}{\partial t}[\alpha \nabla_{x_i} J(x, x_t)] + \frac{\partial}{\partial t}[\beta sign(\nabla_{x_i} J(x, x_t))]] \quad (2.22)$$

By assuming that

$$\|\frac{\partial}{\partial t}[\beta sign(\nabla_{x_i} J(x, x_t))]\| \leq \overline{J}_s \quad (2.23)$$

and

$$\|\frac{\partial}{\partial t}[\alpha \nabla_{x_i} J(x, x_t)]\| \leq \overline{J}, \quad (2.24)$$

where $\overline{J}_s$ and $\overline{J}$ are known positive constants, and choosing

$$u_i = -u_0 sign(s_i) + f_i^k(x_i, \dot{x}_i), \quad (2.25)$$

where

$$u_0 > \overline{M}(\frac{1}{\underline{M}}\overline{f_i} + \overline{J} + \overline{J_s} + \epsilon) \quad (2.26)$$

as the control input for the ith agent, it can be shown that

$$s_i^\top \dot{s}_i < -\epsilon \|s_i\|, \quad (2.27)$$

for any $\epsilon > 0$, which implies that the manifold is reached in finite time. Note that equation 2.23 is not true at the switching instances of the signum function. However,a low pass filter with state, z_i and input $\beta sign(\nabla_{x_i} J(x, x_t))$,

$$\mu \dot{z}_i = -z_i + \beta sign(\nabla_{x_i} J(x, x_t))$$

, can be used to filter out the high frequency signals. μ is a small positive constant. Therefore, the sliding manifold can be redefined as

$$s_{i\_new} = \dot{x}_i + \alpha \nabla_{x_i} J(x, x_t) + z_i. \quad (2.28)$$

Since z_i is bounded, the method derived above could be implemented for this new sliding manifold. Moreover, we have

$$\left\| \frac{\partial}{\partial t}[\beta sign(\nabla_{x_i} J(x, x_t))]_{eq} \right\| = \|\dot{z}_i\| \leq \frac{2\beta}{\mu} \triangleq \overline{J}_s. \tag{2.29}$$

Finally, the controller in (2.25) with s_i replaced with s_{i_new}

$$u_i = -u_0 sign(s_{i\_new}) + f_i^k(x_i, \dot{x}_i). \tag{2.30}$$

with gain u_0 chosen as before, guarantees the occurrence of sliding mode at the redefined manifold s_{i_new} in a finite time. It is important to note that due to this approximation, it may not be possible to ideally recover the exact results that could be obtained for the kinematic model. For example, there may be little difference between the real formation and our desired one. Nevertheless, we still would expect the results to be recovered with reasonable error which is acceptable in many applications.

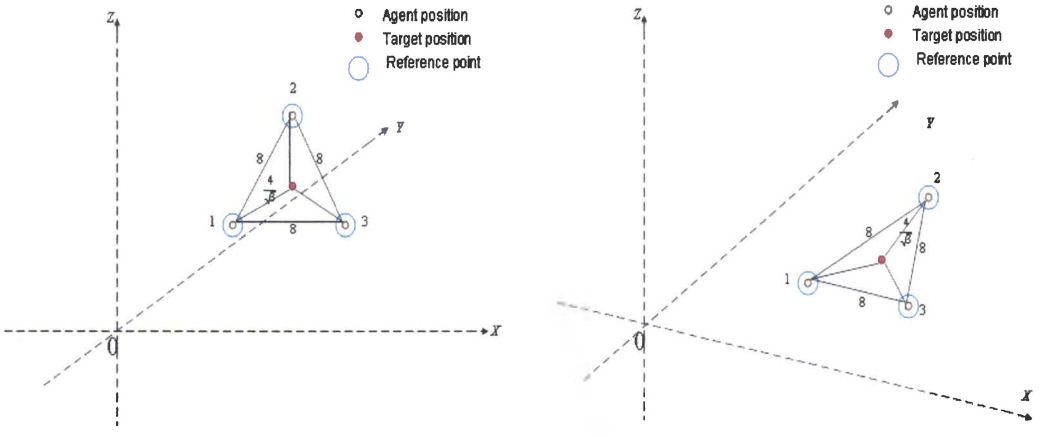
2.5 Results: Matlab Simulations

In this section we present matlab simulation examples to illustrate the control scheme presented above, in particular how to setup the reference frame to achieve formation orientation. We set $n = 3$. Consider a group consisting of $N = 3$ agents with dynamics described by Equation (2.17). Let $M_i = 1$, choose $\underline{M} = 0.5, \overline{M} = 1.5$ and let $f_i^u(x_i, \dot{x}_i) = \sin(0.2t)$ be the unknown uncertainty in the system. For illustrative purposes let's assume target dynamics

$$\begin{aligned} \dot{x}_{t_1} &= 0.3 \\ \dot{x}_{t_2} &= 2\sin(0.2t) \\ \dot{x}_{t_3} &= 1 \end{aligned}$$

We set the initial conditions of agents randomly within an area near the origin, and set the target initially at $[2.5, 2.5, 2.5]$.

Our first objective is to make the three agents form a triangular formation around the target such that the plane of the formation is parallel to the $x-z$ plane at all times. We choose the following formation parameters $\delta_{ij} = 8$ as the distance between agents, and $\delta_{it} = \frac{8}{\sqrt{3}}$ as the distance between the target and an agent. See Figure 2.4(a). For



(a) (b)

Figure 2.4: Formation diagrams: 3D (a) Formation in $x - z$ plane (b) Formation in $x - y$ plane.

simplicity let the positions of the reference points coincide with the positions of the agents as shown. With this configuration, we have the "reference matrix" as

$$\underbrace{h_{ik}}_{1\leq i,k\leq 3} = \begin{bmatrix} 0 & 8 & 8 \\ 8 & 0 & 8 \\ 8 & 8 & 0 \end{bmatrix}.$$

Note that h_{ik} is the relative distance between the position of the ith agent and that of the kth reference point. For instance, since we assumed the positions of the agents to

coincide with the positions of the reference points, the relative distance between agent 1 and the first reference point, $h_{1,1} = 0$. However, the relative distance between agent 1 and the second reference point, $h_{1,2} = 8$. The reference frame for the formation is described by

$$\begin{aligned} \bar{b_1} &= [-4, 0, \frac{-4}{\sqrt{3}}]^\top \\ \bar{b_2} &= [0, 0, \frac{8}{\sqrt{3}}]^\top \\ \bar{b_3} &= [4, 0, \frac{-4}{\sqrt{3}}]^\top. \end{aligned}$$

Relative to the inertial frame, the kth reference point is given by,

$$b_k = x_t + \bar{b_k}, k = 1, ..., 3.$$

Using the potential function in equation(2.16), we select $a_{ij} = 1$,$b_{ij} = 10$ and $c_{ij} = \frac{64}{\ln 10}$ which gives $\delta_{ij} = 8$. Notice that $J(x, x_t)$ has a global minimum, $J(x, x_t) = 0$, when $\|x_i - x_t\| = \delta_{it}$,for all i, $\|x_i - x_j\| = \delta_{ij}$ for all pairs (i, j) and $\|x_i - b_k\| = h_{ik}$ for $i = 1, ..., 3$ and $k = 1, ..., 3$. For the controller parameters, we use $\alpha = 0.01, \beta = 2.0$, $\overline{f}_i = 1$, and $\varepsilon = 1$. For the filter, we chose $\mu = 0.1$. With the choice of the agent initial conditions, we get $\overline{J}$ and $\overline{J}_s$ from (2.23) and (2.24), which produce the value of $u_0 = 138$.

Figures 2.5 (a)-(c) show the tracking, formation and orientation were all achieved as expected.

In the next example, we want the plane of the formation to be parallel to the $x - y$ plane as shown in Figure 2.4 (b). Here the "reference matrix" H and everything else remains as in the previous example except the reference points. We fix the reference

points in the $x-y$ plane as follows:

$$\begin{aligned} \bar{b_1} &= [-4, \frac{-4}{\sqrt{3}}, 0]^\top \\ \bar{b_2} &= [0, \frac{8}{\sqrt{3}}, 0]^\top \\ \bar{b_3} &= [4, \frac{-4}{\sqrt{3}}, 0]^\top. \end{aligned}$$

As before, Figures 2.5 (d)-(f) show expected tracking, formation and orientation. Thus you can always define the plane of orientation by fixing the reference points. This procedure can be applied for any n.

In the following examples we set $n = 2$ and $N = 2$. We choose the formation parameters $\delta_{ij} = 10$ and $\delta_{it} = 5$. The target dynamics are choosen as

$$\begin{aligned} \dot{x}_{t_1} &= 1 \\ \dot{x}_{t_2} &= 1. \end{aligned}$$

We set the initial conditions of agents randomly within an area near the origin, and set the target initially at $[2, 2]$. First we want the formation orientation (the angle between the line through the position of the first agent and the position of the second agent on one hand and the x-axis) to be 135^o. See Figure 2.6 (a). Thus we have

$$\underbrace{h_{ik}}_{1 \le i,k \le 2} = \begin{bmatrix} 0 & 10 \\ 10 & 0 \end{bmatrix},$$

and the reference frame for the formation described by

$$\begin{aligned} \bar{b_1} &= [-5\sqrt{2}, 5\sqrt{2}]^\top \\ \bar{b_2} &= [-5\sqrt{2}, -5\sqrt{2}]^\top. \end{aligned} \tag{2.31}$$

In our final example we maintain all parameters but the orientation, we set it to 45^o as shown in Figure 2.6(b). This gives

$$\underbrace{h_{ik}}_{1\leq i,k\leq 2} = \begin{bmatrix} 0 & 10 \\ 10 & 0 \end{bmatrix}$$

and

$$\begin{aligned} \bar{b_1} &= [5\sqrt{2}, 5\sqrt{2}]^\top \\ \bar{b_2} &= [-5\sqrt{2}, -5\sqrt{2}]^\top. \end{aligned} \tag{2.32}$$

As observed in Figure 2.7 tracking, formation and orientation were all as expected in both cases.

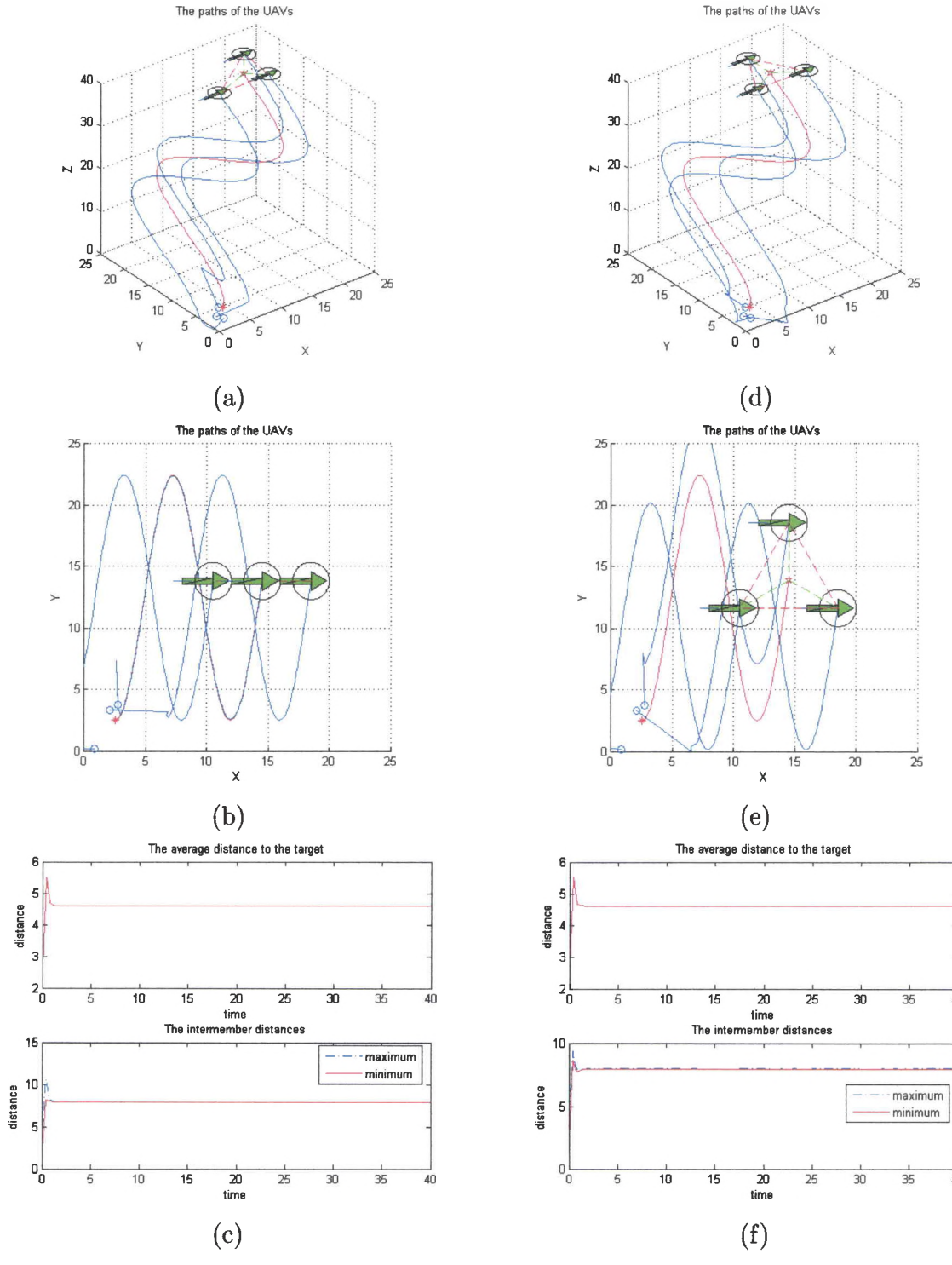
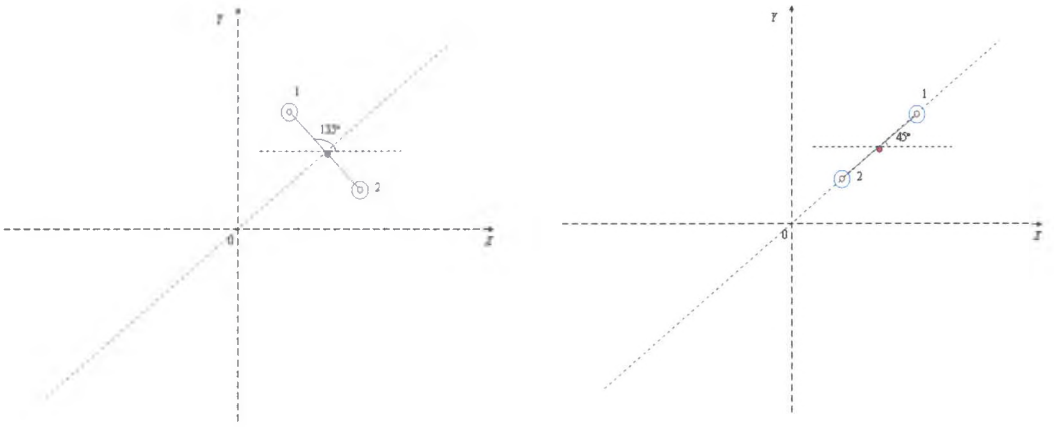


Figure 2.5: Simulation results: (a) Formation in $x-z$ plane (b) Looking down the $x-y$ plane (c) Formation distance (d) Formation in $x-y$ plane (e) Looking down the $x-y$ plane (d) Formation distance.



(a) (b)

Figure 2.6: Formation diagrams: 2D (a) Formation orientation: 135^o (b) Formation orientation: 45^o.

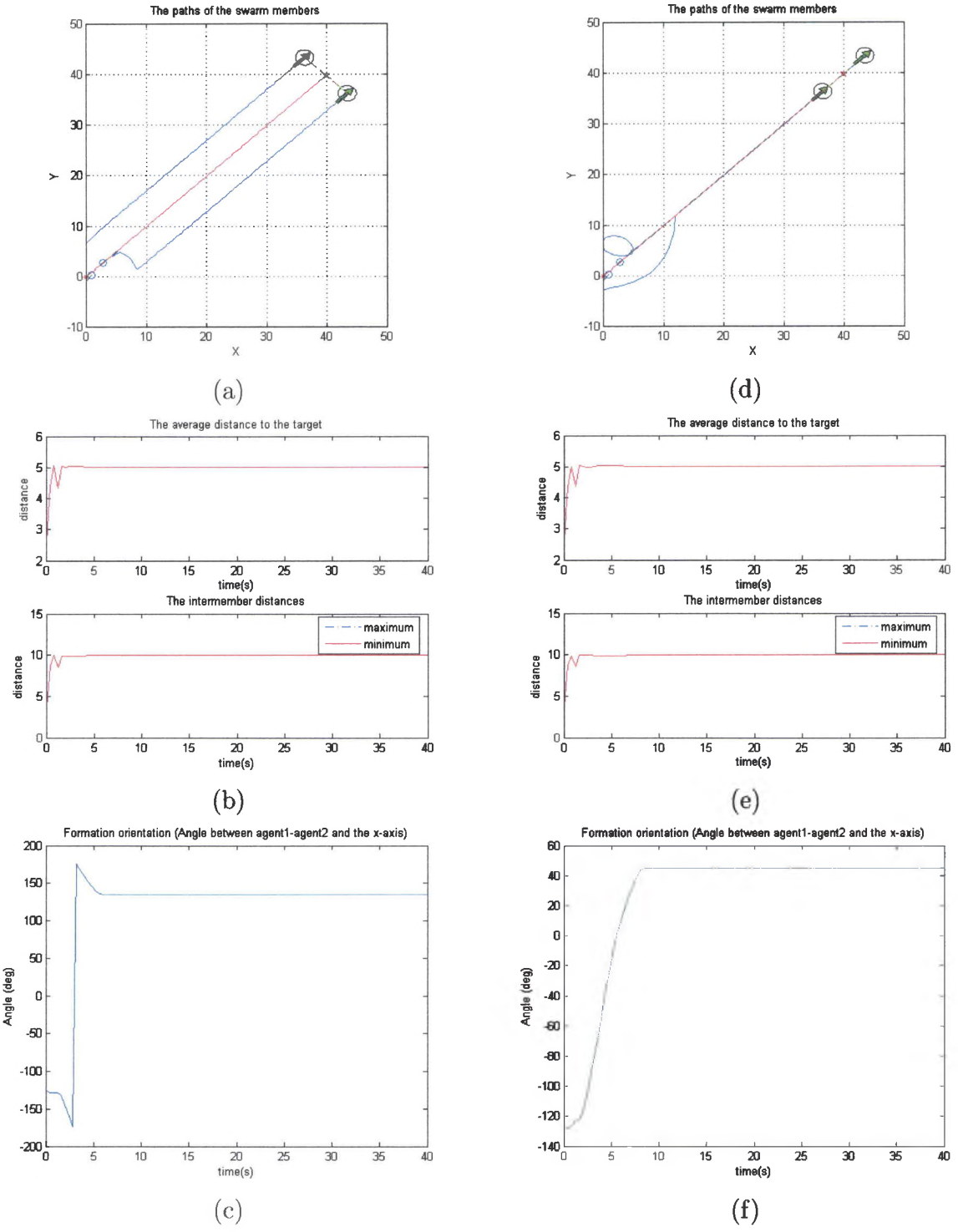


Figure 2.7: Simulation results: (a) Path of agents for orientation = 135^o (b) Formation distances (c) Formation orientation (d) Path of agents for orientation = 45^o (e)Formation distances (d) Formation orientation.

CHAPTER 3

CONTROLLER IMPLEMENTATION

This chapter describes the software implementation of the coordination stratagy discussed in the previous chapter. We first describe the components of the control software and then a description of a custom graphical user interface developed for the formation control algorithm is also given.

3.1 Formation Control Program

The formation control code is the C/C++ implementation of the decentralized control stratagy discussed in Chapter 2. The program runs on a Linux laptop and communicates with the two UAVs through a wireless LAN. Each UAV is controlled by a separate process. The processes communicate with the UAVs through a Wireless Local Area Network (WLAN) and interprocess communication is through shared memory. A single process runs as a virtual target. We assume negligible communication delay in communications between the processes and the UAVs. The motion of the virtual target defines the motion of the formation. The former is defined by the user. Position update is done via UDP across the WLAN to the UAVs. The process running the virtual target (Process3) computes its position at each time step and updates the shared memory. Process1 and Process2 request position update from UAV1

and UAV2 respectively, update the shared memory, compute the next positions and transmit that data to UAV1 and UAV2 respectively via the WLAN. A representation of the program is shown in Figure (3.1).

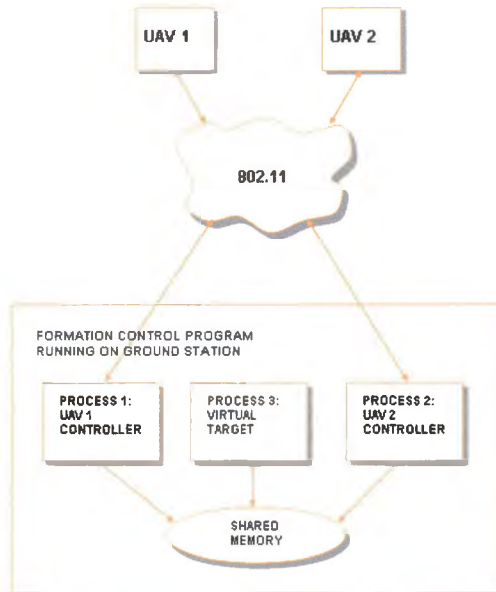


Figure 3.1: A block diagram representation of the formation control code

3.2 GUI Development

A graphical user interface (GUI) was essential for the formation control code to serve as the front end to the user both during simulation in the virtual environment and the actual experiment. Using C/C++ and the non-comercial version of the Qt3 software(a C++ GUI Application Programming Interface), a custom GUI was designed to serve that purpose. The GUI had to meet several objectives. The user must be able to:

- Enter network/connection parameters
- Enter formation parameters
- Define formation trajectory
- Run and time simulation/experiment
- Plot trajectory of each UAV
- Detect unsafe scenarios and abort mission if necessary
- Collect data for each UAV

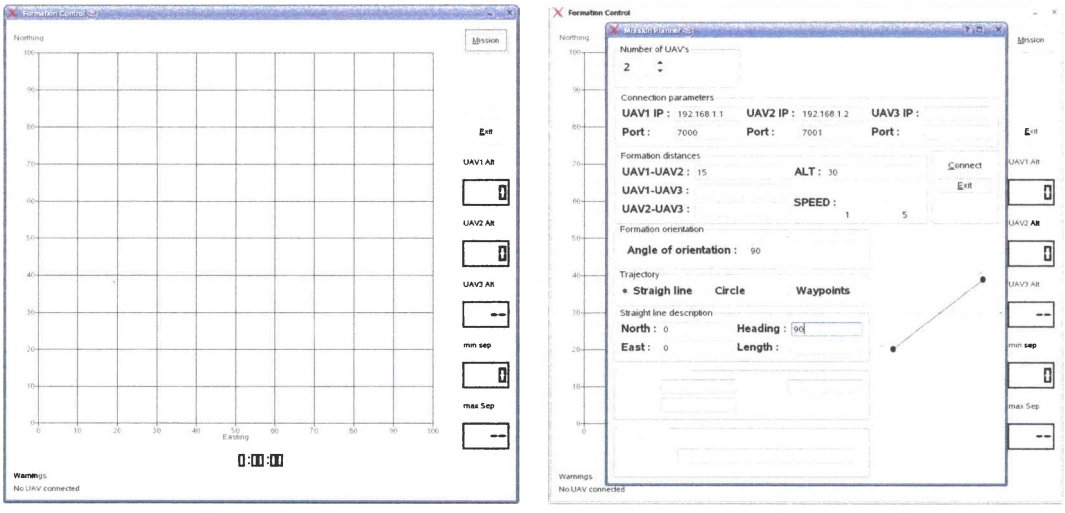
All of these objectives were realized as shown in the Figure 3.2 and Figure 3.3 . Figure 3.2 (a) shows the main window for the program. The user clicks on the mission button to type in configuration information. First, the IP's and the port numbers of all the UAV's are typed in. As shown in Figure 3.2 (b),the controlling processes use the IP's and the port numbers to establish communications with the UAVs. Next, the required distances between the UAV's are typed in as well as the altitude. Since the application requires all the UAV's to be in the same plane, the altitude applies to all of them. The user can then use the slide to set the speed of the virtual target and then go on to set the formation orientation with respect to the x-axis. The desired trajectory is then selected. The user can currently choose from two trajectories, straight line or circular. We have made provisions in the GUI for future waypoints option.[3] The final details of the trajectory are then specified. Once all acceptable data are entered, the user can click on the "connect" button and the controlling

[3]The formation control algorithm currently supports continuous formation trajectory. Algorithms that generate a continous trajectory from a set of discrete waypoints are currently being investigated.

programs will then be ready to run. The running mode of the GUI is shown in Figure 3.3 (a). The current positions of the UAV's are indicated with a red "plus" symbol and the circle around the "plus" represents the boundary of a "safe region" around the UAV. The circle has a diameter of $6m$ and represents the closest distance another UAV can approach before the safety feature is initiated. The safety feature will be discussed in the next section. This means the smallest posible formation distance is $12.0m$ to prevent the system from going into "safe mode". Note that these limits can always be changed. The position of the virtual target has the same representation as the UAV's but colored in black. The desired trajectory is drawn in green and the tracks of the UAV's are in magenta and blue. The scale of the local map is in meters. The units of the formation parameters should also be in meters. Once the controlling programs are running, the main window displays the track and position of each UAV on the map, the altitude of each UAV and the minimum and maximum separation of each UAV from each other as well as simulation/running time all in real time.

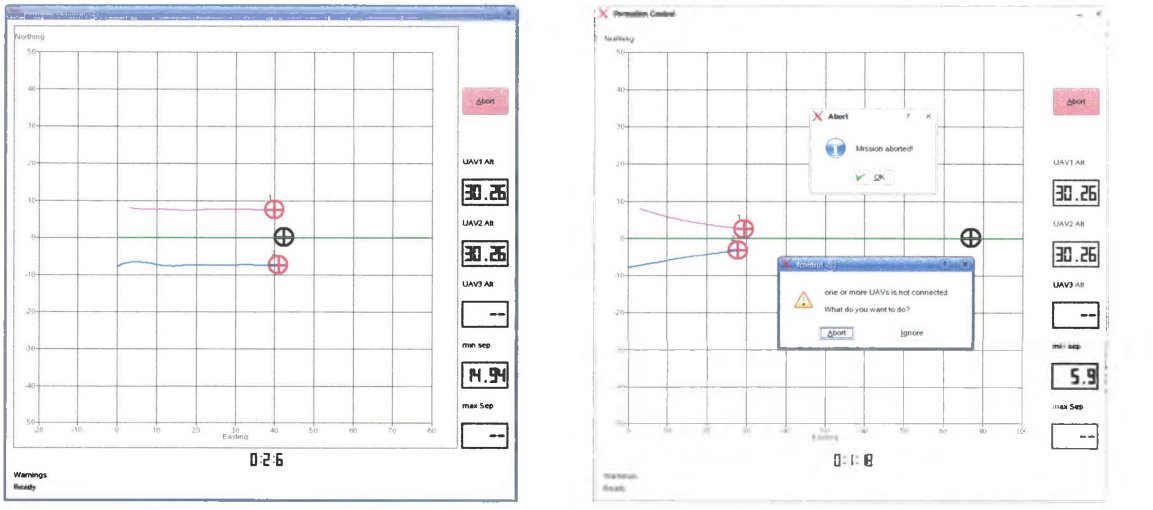
3.3 Safety Issues

Although careful choice of potential function and initial conditions may prevent collision between the agents, disturbances like strong winds and communications delay may cause the agents to deviate from the desired formation distances. To avoid such a scenario, a safety feature is built into the GUI to warn the user of potentially dangerous manuevers and automatically aborts the mission. This feature can be seen in Figure 3.3 (b). When the safety feature is initiated, the onboard autopilot of each UAV is given full control and each is flown to a failsafe position awaiting further manual instruction from a ground operator through a Futaba radio.



(a) Main window (b) Mission window

Figure 3.2: Formation Control software front-end GUI.



(a) Running mode (b) GUI safety feature automatically enables.

Figure 3.3: GUI running mode

CHAPTER 4

SIMULATION IN VIRTUAL ENVIRONMENT

4.1 Virtual Environment

Development of the formation control algorithm was performed in a virtual environment using simulated robots. The open source tools Player/Stage/Gazebo provide an excellent real-time robot simulation platform especially for multi-robot applications and sensor network systems. These tools simplify controller development by providing a well documented C++ API that relieves the developer from the need to write low-level interface and communication code for real-world robots and sensors. Player/Stage/Gazebo is the choice for most researchers in recent days. The package was first developed by the University of Southern California (USC) Robotic Research Lab in 1999. It has since then been adapted, modified and extended by researchers around the world. USC defines each component as follows:

- Player: Player is a device server that provides a powerful, flexible interface to a variety of sensors and actuators (e.g., robots). Because Player uses a client/server model, robot control programs can be written in any programming language and can execute on any computer with network connectivity to the robot. In addition, Player supports multiple concurrent client connections to

devices, creating new possibilities for distributed and collaborative sensing and control.

- Stage: Stage is a scaleable multiple robot simulator; it simulates a population of mobile robots moving in and sensing a two- dimensional bitmapped environment, controlled through Player. Stage provides virtual Player robots which interact with simulated rather than physical devices. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry.

- Gazebo: Gazebo complements stage's simulation capabilities, focusing on higher fidelity dynamics based simulation in a 3D environment. Player is still used as an interface to the robot, making simulations in Gazebo, Stage and physical robots an easy task to accomplish.

4.2 Gazebo Simulations

Gazebo and Player were the simulation tools used in this research. This section describes two simulation scenarios to verify the formation control algorithm. The OsuHeli model was chosen as our helicopter model in the Gazebo world. This model was derived, by the Ohio State University Cooperative Center of Control Science, from the original AvatarHeli model which simulates the USC AVATAR helicopter, a Bergen Industrial Twin RC helicopter. The AvatarHeli model responds only to velocity commands but the OsuHeli model responds to position commands, hence it is suitable for this particular algorithm. The GarminGPS model which models a Garmin EOM GPS was the only position measuring sensor on the helicopters. Player provided the interface between the formation control code and the Gazebo models. It

must be noted that all the real-time simulations done here were done over a network (the Gazebo models were run on a separate computer on the same network as the computer running the formation control program) so as to subject the controller to communication delays and all the other factors that we may face during the actual experiment. We named our helicopter models Helo1 and Helo2.

4.2.1 Straight Line Trajectory

Here we move on to verify the stratagy proposed in Chapter 2 performing simulations in virtual environment. In the first simulation we choose formation parameters as follows:

- Formation distance : $20.0m$
- Formation orientation: 135^o to the x-axis (anti-clockwise)
- Formation trajectory : straight line 045^o heading.

We initialize the helicopters houvering at $[144, 160, 30.0]^\top$ for Helo1 and $[154, 140, 30.0]^\top$ for Helo2 in the world coordinates. The second simulation has the following parameters:

- Formation distance : $20.0m$
- Formation orientation: 0^o to the x-axis
- Formation trajectory : straight line 090^o heading(along the x-axis).

The initial position of the helicopters were, $[157, 148, 30]^\top$ for Helo1 and $[144, 150, 30]^\top$ for Helo2. Figure 4.1 shows snapshot of paths of the helicopters for the first case and Figure 4.2 for the other case. The controller performance for both cases can also be

seen in Figure 4.3. An interesting oberservation can be made in the results of the simulation. It can be seen in Figure 4.3 that we did not obtain perfect formation, there seems to be performance trade off depending on formation orientation and/or formation trajectory. It was observed that situations where we obtained pretty good formation distance, we fell short on formation orientation and vise vesa. The formation trajectory was however followed closely. The reason for this behavior is not readily apparent, further studies need to be conducted.

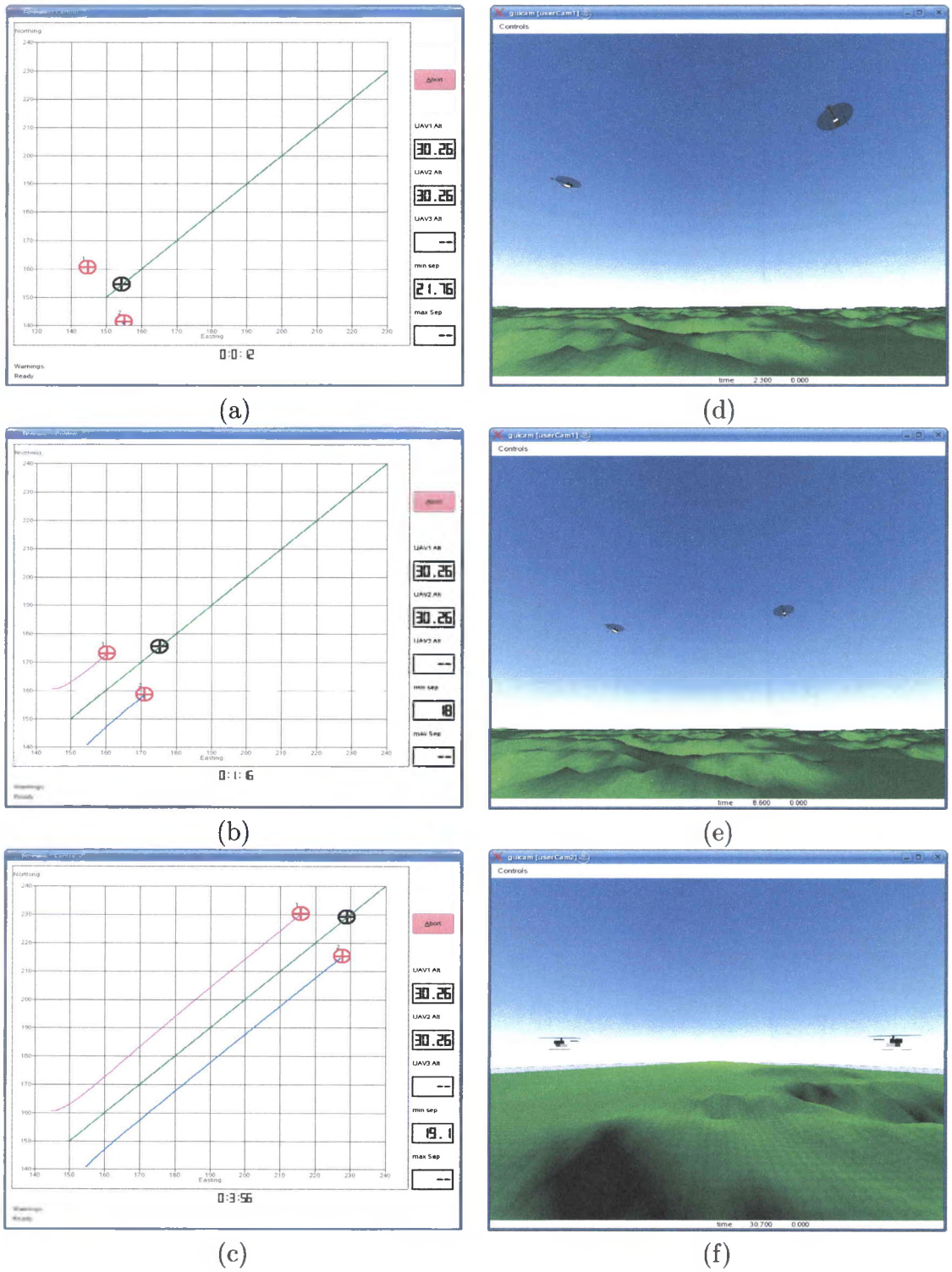


Figure 4.1: Simulation results case 1: $\delta_{ij} = 20$, $\delta_{it} = 10$, orientation = 135^o or -45^o. (a)-(c) GUI snapshots of the Gazebo simulation. (d)-(f) Gazebo snapshots.

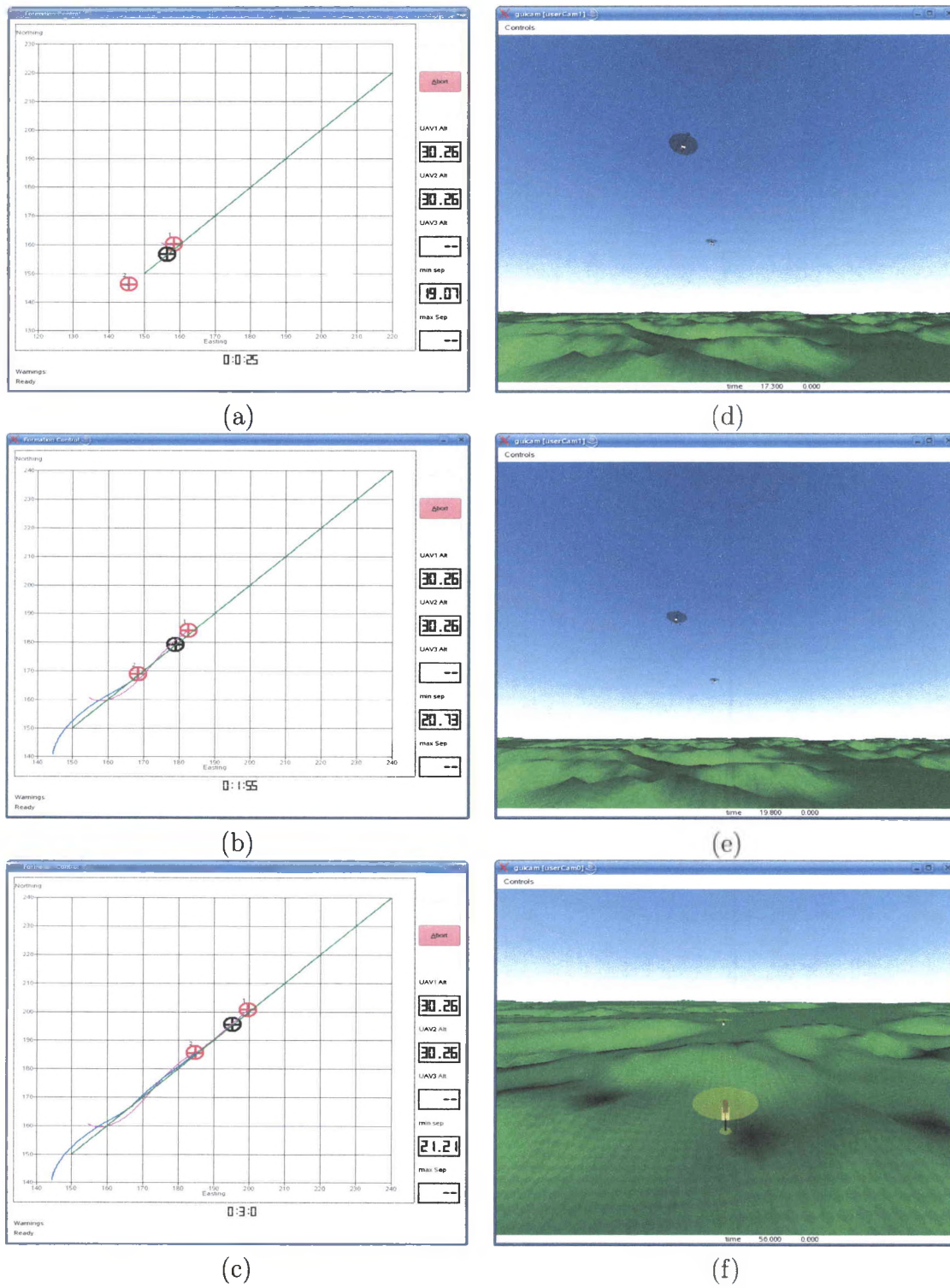


Figure 4.2: Simulation results case 1: $\delta_{ij} = 20$, $\delta_{it} = 10$, orientation $= 45^o$. (a)-(c) GUI snapshots of the Gazebo simulation. (d)-(f) Gazebo snapshots

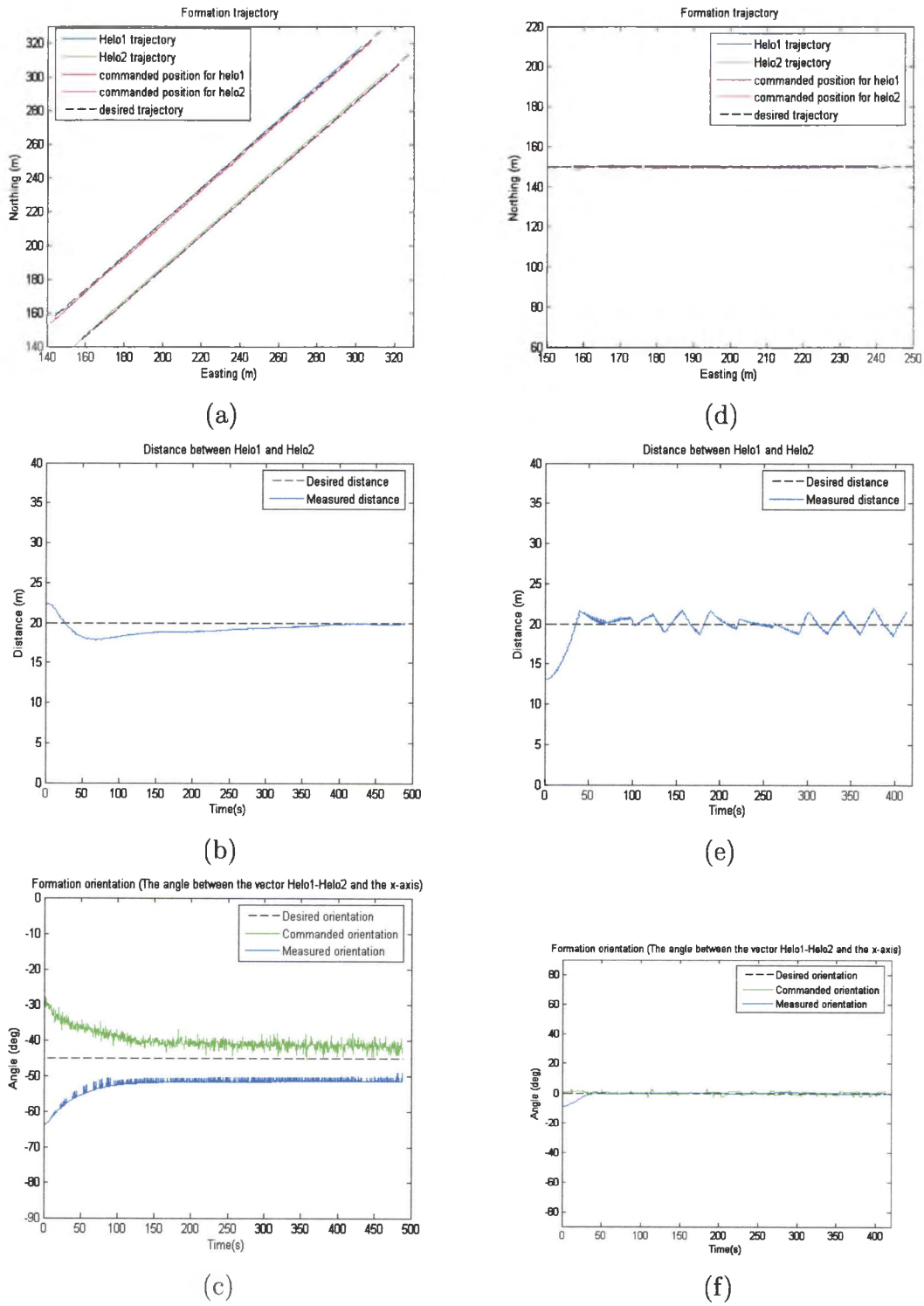


Figure 4.3: Controller performance:(a),(b) Formation trajectories for cases 1 and 2, respectively. (c)-(d) Formation distances (e)-(f) Formation orientation.

CHAPTER 5

IMPLEMENTATION RESULTS

In this chapter we show experiments supporting the theory discussed in chapter 2. These experiments are performed on real-time helicopters named "Helo 1" and "Helo 2" (see Figure 5.1.) However, at the time of this thesis, flight restrictions at the WPAFB made it impossible to perform actual flight experiments. The experiments shown here are ground tests performed on the helicopters. First, we describe the experiment setup. Next, we perform a ground test on the helicopers and then repeat an identical test in the virtual environment and compare the performance of the controller.

5.1 Testbed Setup

The two helicopters shown in Figure 5.1 were developed by Bergen RC and serve as a testbed for research work at the WPAFRL Sensor Directorate. These small research and development platforms share a common architecture and foundation software. Each helicopter has an onboard Autonomous Flight Control System (AFCS) from Rotomotion. The Integrated Flight Controller comprises the Atittude and Heading Reference System (AHRS), GPS receiver, servo controller, safety system and an

802.11 data telemetry transceiver. Figure 5.2 (a) shows the configuration of the entire system and Figure 5.2 (b) shows configuration of the Rotomotion AFCS. The AHRS system is composed of three accelerometers to measure movement along the three linear degrees of freedom axis, three gyros to measure rotation along the three rotation axis and three magnetometers to measure relative magnetic fields and act as a compass to find north. The information gathered from the AHRS and GPS data are processed through a Kalman filter to provide state data for control purposes. The AFCS translates the request of the operator into helicopter movements. It has two modes of operation namely, manual and autopilot. In manual mode, the AFCS receives control input from a test pilot transmitter via the Rotomotion modified Futaba 9-channel radio receiver. It has been modified to output pulse code modulated (PCM) signals to the AFCS. In autopilot mode, the AFCS puts the helicopter in hover mode ready to receive waypoints from the ground station. The waypoints are position and heading commands and can be defined as relative displacement to the current position in either the body frame or the north-east-down (ned) coordinate system. It can also be defined as displacement in absolute coordinates based on the ned system or latitude longitude and altitude (LLH) coordinates.

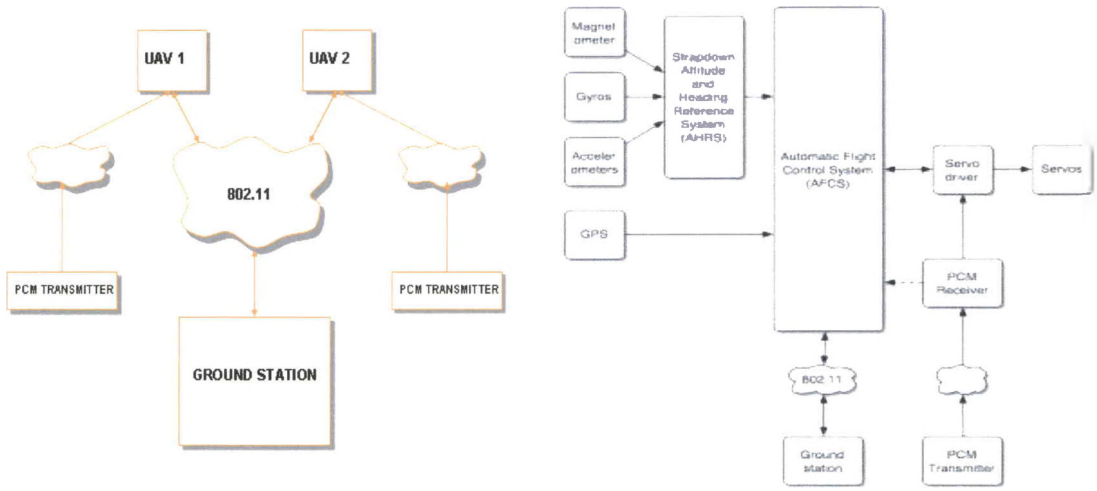
A Pentium 4 laptop running a Linux (kernel v 2.6.15) Operating System and Rotomotion telemetry software (see Figure 5.3) served as the ground station. The ground station API software allows the user to write custom client-server control software for the helicopters. All user control and utility software reside on the ground station computer. Communication between the ground station and all UAV's is through a Wireless LAN. The onboard sensors and communication device (see Figure 5.4) are all powered by an onboard $12V$ battery.

(a) Helo 1 (b) Helo 2

Figure 5.1: Bergen helicopters that were used in the experiments.

5.2 Ground Test: Real World

The ground test was performed with Helo1 and Helo2 set up as shown in Figure 5.5(b). First a data network had to be setup between the helicopters and the ground station. The network configuratation is shown in Figure 5.6. The network consists of the AFCS from both helicopters, four wireless access points, a router and the ground station computer (see Figure 5.7). A router was required because both AFCS were on different IP subnets. The helicopters were placed on carts and initialized at random positions near the origin of the local tangent plane and then pushed on the pavement in a predifined formation. Again, the idea was to observe the behaviour of the controller in a scenario where current positions of the helicopters are known but commanded positions to them are completely ignored. This is an unsual scenario but it provides an intuitive way to be able to reproduce the ground test in Gazebo



(a) System Configuration (b) AFCS configuration.

Figure 5.2: (a) Top level system configuration (b) Rotomotion flight control system configuration .

simulation and observe if similar controller behaviour will be obtained.

The formation control software was given the following formation parameters:

- Distance between carts (i.e., formation distance): $15.0m$
- Formation orientation: 90^o to the x-axis
- Formation trajectory : along the x-axis (straight line 090^o heading)

5.3 Ground Test: Virtual Environment

In Gazebo world, each helicopter was placed on a car chasis (carts) see, Figure 5.5 (a). The Gazebo carts were made to trace the path of the ground test carts. In effect, we are reproducing the ground test in Gazebo so we can compare controller behavior in both the real world experiment and the Gazebo simulation. Figure 5.8 shows snapshots of the simulation in Gazebo.

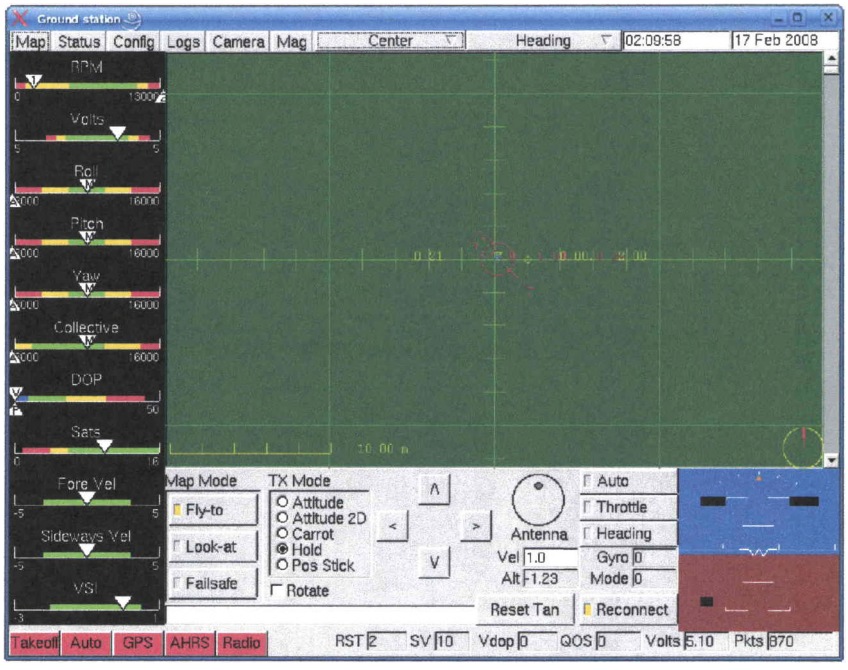
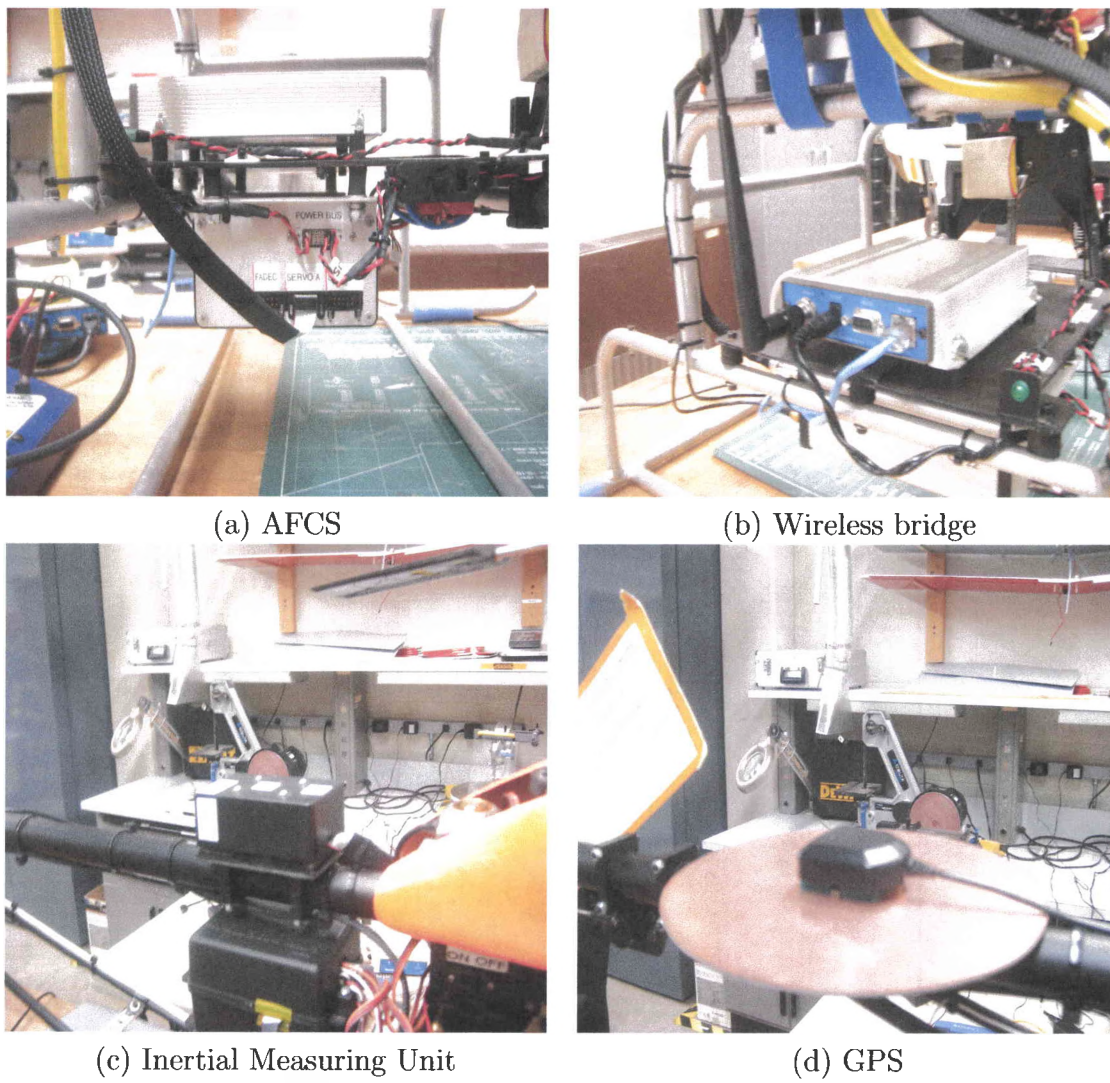


Figure 5.3: Rotomotion ground station telemetry program GUI.

Figure 5.8 (c) and (d) show the trajectories of the Gazebo simulation and the experiment respectively in the formation control GUI. Figure 5.9 shows the controller performance in both the virtual environment and the actual experiment. It was observed that in both cases the controller tries to correct the path of the carts.

As it was noted earlier on, Gazebo presents high fidelity models for its robots addition to the fact that all simulations were conducted over a network, verification of the controller in the virtual world, most of the time, is the final step in controller implementation.



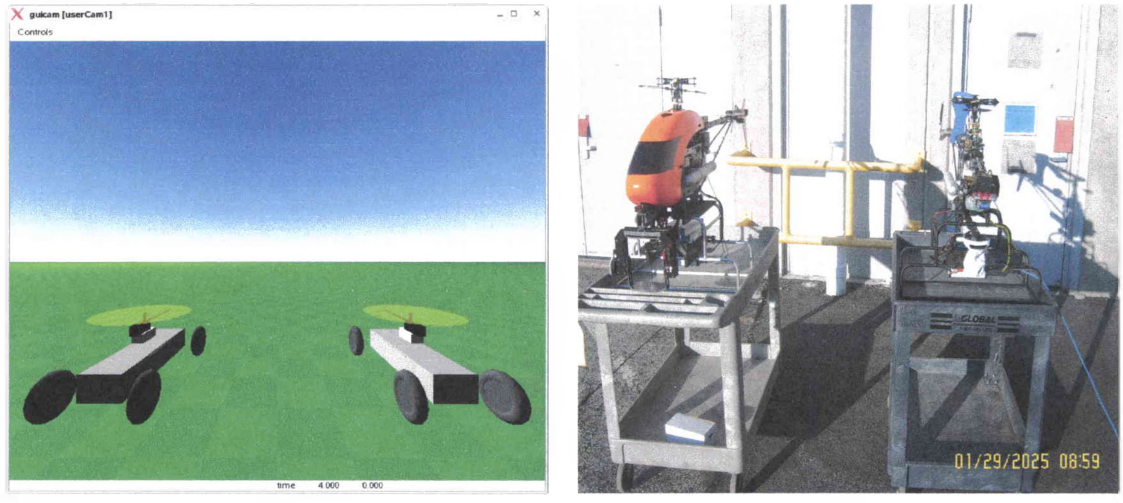
(a) AFCS

(b) Wireless bridge

(c) Inertial Measuring Unit

(d) GPS

Figure 5.4: Sensors and devices on the helicopter.



(a) (b)

Figure 5.5: (a)Helicopter-cart setup unit in Gazebo (b)Helicopter-cart setup unit for actual experiment.

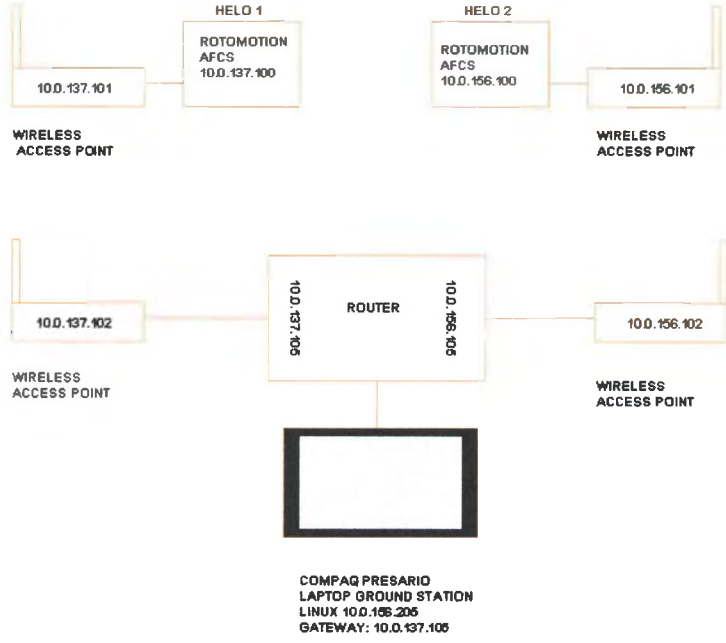


Figure 5.6: Helicopter system network configuration.

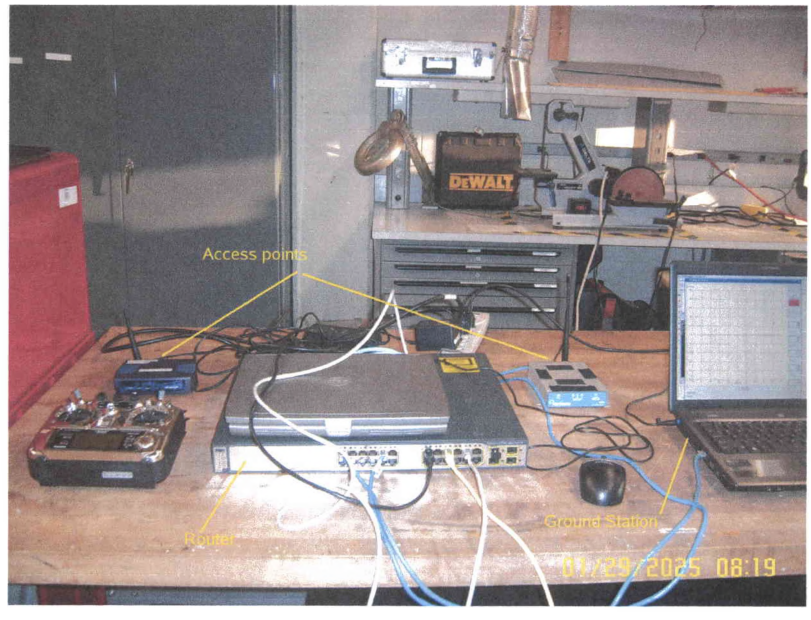


Figure 5.7: Helicopter network setup.

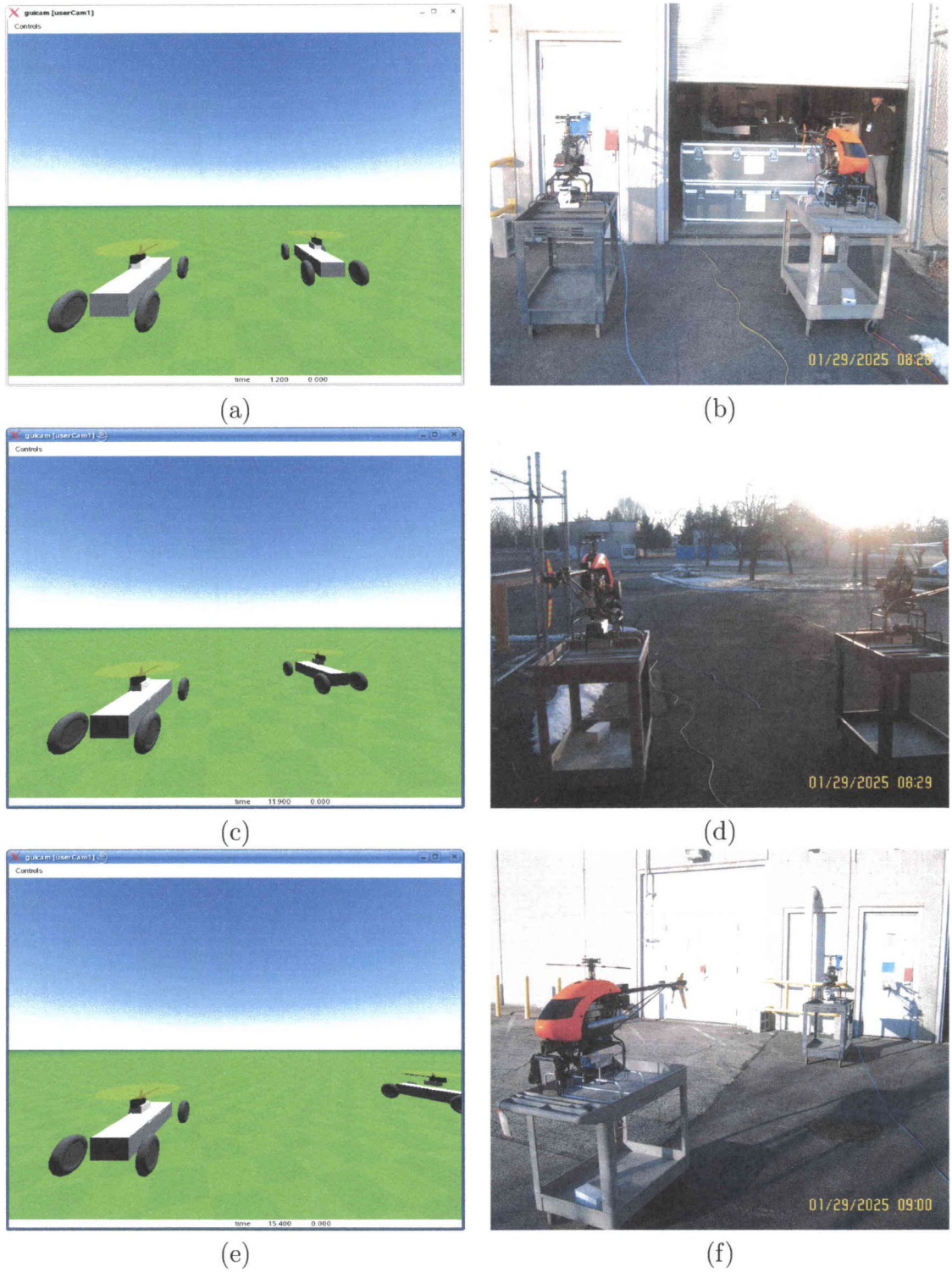


Figure 5.8: Snapshots of the ground test (a) Gazebo snapshots (b) Snapshots from experiment.

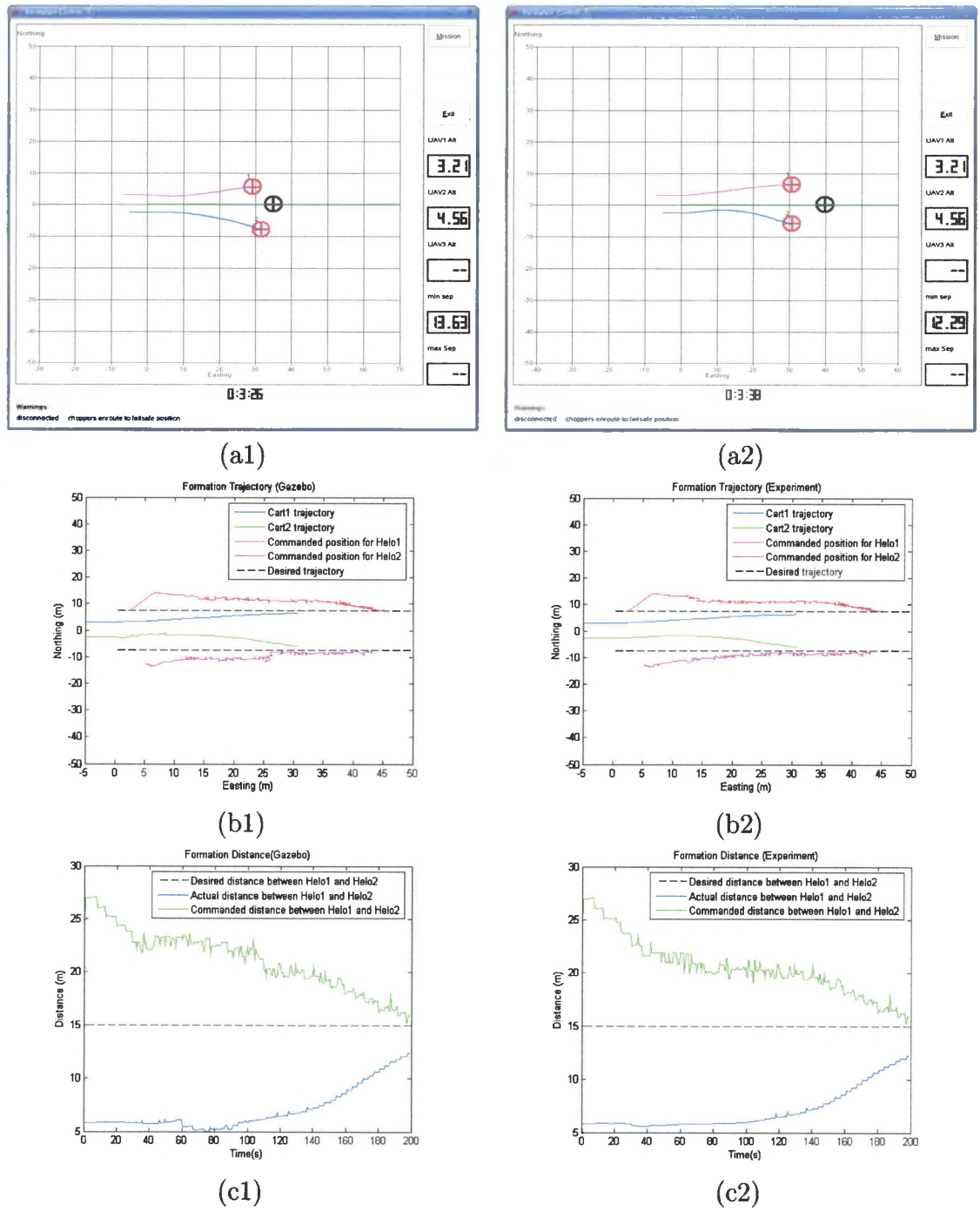


Figure 5.9: Controller performance in ground test. (a1)-(c1) Gazebo simulation (a2)-(c2) Actual experiment.

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

6.1 Conclusions

The primary purpose of this research was to implement the coordination tracking stratagy developed by the research group at the Non-linear Systems and Controls Laboratory at the University of Dayton on exprimental helicopters in cordination with the Wright Patterson Air Force Reasearch Lab. The original work was extended to include formation orientation. It was shown that stability results obtained for the original work was preserved. Matlab simulations results were provided to support the extension. The algorithm was developed in the Player/Gazebo virtual environment and later implemented on Bergen RC helicopters. Results obtained was similar to that proposed in theory and show that the stratagy is practically implementable.

6.2 Future Research

The control technique was extensively verified in the Player/Gazebo virtual environment and as in most cases, the code used in the simulation was essentially identical to the implementation code. Though only ground testing was performed in the verification process of the control strataty on the hardware, the results obtained are

strong evidence of correctness of the procedure. Nevertheless, there is always room for improvements. First of all, a point mass motion was assumed for the helicopters in the analysis of this algorithm. The algorithm ignores the dynamics of the helicopter and makes it impossible to control anything else apart from position and thus can only be used to generate reference trajectory for the autopilot. In the context of the application requirements of this thesis, this is acceptable since the autopilot takes over the other system states. However, in more complex applications, this may not be acceptable. Therefore, the procedure may be extended to agents with more realistic system dynamics, for instance the six degree of fredom model of the helicopter. Again, the formation control procedure can still be extended to include formation switching and formation rotation. Formation switching is where the agents switch from one formation to the other while preserving stability. Formation rotation will be a direct extension to the formation orientation procedure proposed in Chapter 2. Making the reference frame time dependent would result in formation rotation but extra theoretical analysis needs to be done to ensure formation stability. Although necessay steps was taken to avoid collision amongst the agents, analytical work need to be done to guarantee collision avoidance. Finally, the procedure currently supports only continuous target trajectories, another area of future research will be to propose an algorithm which will generate a continuous trajectory through a set of discrete waypoints which will serve as formation trajectory.

CHAPTER 7

APPENDIX I
POSITION REFERENCE CODE

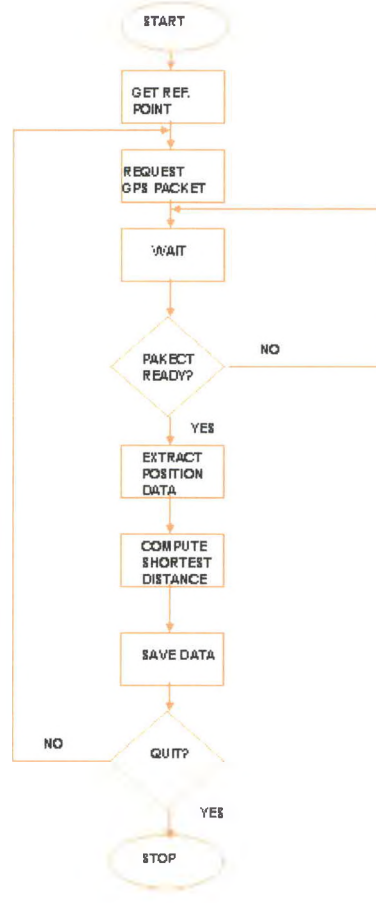


Figure 7.1: Flowchat for the position reference code.

7.1 posref.cpp

```
/*******************************posref.cpp*************************************
 *
 *  This program calculates the shortest distance between a UAV in flight
 *  and a reference position .
 *
 ******************************************************************************
 *  Created by Boakye Dankwa
 *  ECE Department, University of Dayton.
 *
 *  University of Dayton in collaboration with
 *  Air Force Research Laboratory
 *  Sensors Directorate
 *
 *  Version 1.0
 *  Created: June 13, 2006
 *
 ******************************************************************************/

#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <unistd.h>
#include <time.h>
#include <ctype.h>
#include <iostream>
#include <signal.h>
#include <include/macros.h>
#include <include/timer.h>
#include <include/byteorder.h>
#include <getoptions/getoptions.h>
#include <state/messages.h>
#include <state/Server.h>
#include <mat/Conversions.h>
#include <mat/Nav.h>

#include <flightgps.h>

using namespace libstate;
using namespace libmat;
using namespace util;
using namespace std;

/**********************
 Command promt help
**********************/
static int help( void )
{
cerr <<
```

```
"\nThis program calculates the distance between the UAV in flight and a reference\n"
"position and saves the output in a text file. \n"
"\nUsage: posref -[options] -- lat lon alt [filename]\n"
"\n"
"Options:\n"
"       -h | --help        This help\n"
"       -m | --meters                  meters --units of output (default)\n"
"       -f | --feet                    feet\n"
"       -y | --yards                   yards\n"
"       -M | --miles                   miles\n"
"       -N | --nautical                nautical miles\n"
"\n"
"the other arguments(reference coordinates) are:\n\n"
"     lat     : latitude       input format : dddmmss(N/S) or +/-ddd.dddd (deg)\n"
"     lon     : longitude      input format : dddmmss(E/W) or +/-ddd.dddd (deg)\n"
"     alt     : altitude       input format : hhh.hhhh   (meters)\n"
"     filename  : name of the output file. Default is positionlog\n"

<< endl;

return -10;
}
int quit = 0;

/**********************
 file stream for logging
**********************/
FILE * pstnLogFile;

/**********************
 name of log file
**********************/
char *pstnlog = "positionlog";

/*************************************************************************
 *      This reads the current position from the AFCS and calculates
 *      the distance between a reference position and the current position.
 *              inputs: current position of UAV
 *              output: none
 *************************************************************************/
void distanceFromRef(const Vector<3> &refCoordECEF);

/*************************************************************************
 *      This computes the distance between two ECEF position
 *              inputs: reference ECEF position of UAV,
 *                      current ECEF position of UAV
 *              output: none
 *************************************************************************/
void findDistance( const Vector<3> &position1, const Vector<3> &position2);

/*************************************************************************
```

```
 *      This function logs the distance and position of the UAV.
 *              inputs: reference ECEF position of UAV,
 *                      current ECEF position of UAV
 *              output: none
 ************************************************************************/
void saveResults(const Vector<3> &pstn1, const Vector<3> &pstn2);
/**********************
 UTILITY FUNCTIONS
**********************/
/************************************************************************
 *      This function converts LLH(deg,min,sec) to decimal degrees.
 *              input: (deg,mim,sec)
 *              output: decinal degree
 ************************************************************************/
double llh2deg(char* argv);
/************************************************************************
 *      This function initializes the log file.
 *              input: none
 *              output: none
 ************************************************************************/
void initLogFile();
/************************************************************************
 *      This function returns the unit of results.
 *              input: none
 *              output: unit(feet,meters,miles,nautical miles)
 ************************************************************************/
const char* units();
/*******************************
 Exit handler
*******************************/
void cleanup(int);
/*******************************
 Distance between two positions.
*******************************/
double dist   = 0.0;
/***************************************
 * Angle between the UAV and the
 * horizontal at the reference position.
 ***************************************/
double alpha  = 0.0;

static int meters   = 0;
static int feet     = 0;
static int yards    = 0;
static int miles    = 0;
static int nautical = 0;

const char me[10] = "meters";
const char fe[10] = "feet";
const char ya[10] = "yards";
const char Mi[10] = "miles";
```

```
const char Na[20] = "nautical miles";

int main( int argc,char ** argv  )
{
         const char *  program = argv[0];

     int rc = getoptions( &argc, &argv,
                                "h|?|help&",   help,
                                "m|meters+",       &meters,
                                "f|feet+",         &feet,
                                "y|yards+",         &yards,
                                "M|miles+",        &miles,
                                "N|nautical+",     &nautical,
                                0
                                );

 if( rc < 0 || (argc !=3 && argc != 4 ))
    {
cerr << "Usage: " << program << " -[options] --lat lon alt [filename]" << endl;
return EXIT_FAILURE;
     }

         if (argc == 4 )
               {
                        pstnlog = argv[3];
                        }
          // use typed coordinates as reference position
               int nlatdigit = strlen(argv[0]);
               int nlondigit = strlen(argv[1]);
               int naltdigit = strlen(argv[2]);

              char latcardinalPoint = *(argv[0] + (nlatdigit -1));
              char loncardinalPoint = *(argv[1] + (nlondigit -1));
              char altcardinalPoint = *(argv[2] + (naltdigit -1));
              if ( isalpha(latcardinalPoint)&&(latcardinalPoint !='n')
                        &&(latcardinalPoint !='N')
                        && (latcardinalPoint !='s')
                        && (latcardinalPoint !='S'))
                        {
                          cerr << " wrong cardinal point for lat. "<<endl;
                          exit(EXIT_FAILURE);
                               }
              if ( isalpha(loncardinalPoint)&&(loncardinalPoint !='e')
                        &&(loncardinalPoint !='E')
                        && (loncardinalPoint !='w')
                        && (loncardinalPoint !='W'))
                        {
                          cerr << " wrong cardinal point for lon. " <<endl;
                          exit(EXIT_FAILURE);
                               }
              if( isalpha(altcardinalPoint))
```

```
                    {
                      cerr << " Altitude shouldn't have cardinal point" << endl;
                      exit(EXIT_FAILURE);
                            }

            double lat       = llh2deg( argv[0] ) * C_DEG2RAD;
            double lon = llh2deg( argv[1] ) * C_DEG2RAD;
            double alt = atof( argv[2] );

            const Vector<3> llhA( lat, lon, alt);
            const Vector<3> refCoord(llh2ECEF(llhA));

            signal(SIGINT,cleanup);
            distanceFromRef(refCoord);

return 0;

}
/*************************************************************************
 *      This reads the current position from the AFCS and calculates
 *      the distance between a reference position and the current position.
 *              inputs: current position of UAV
 *              output: none
 *************************************************************************/
void distanceFromRef(const Vector<3> & refPstnECEF)
{
     const char *    server_host = "10.255.0.7";
     int               server_port = 2002;

     Vector<3>  curPstnECEF(0.0,0.0,0.0);
     Vector<3>  curPstnLLH(0.0,0.0,0.0);

     flightgps       gps;

     printf("\nINITIALIZING LOGFILE ...");
        initLogFile();

     printf("\nCONNECTING TO SERVER...");
        gps.connect(server_host,server_port);

     for (;;){

           if( !gps.takeReadings() )
                 exit(EXIT_FAILURE);

              const msg_gps_t  * currentPosition = (const
              msg_gps_t*)(gps.getMessege(GPS_STATE));

              curPstnECEF[0] = currentPosition->raw_pos_n;
              curPstnECEF[1] = currentPosition->raw_pos_e;
              curPstnECEF[2] = currentPosition->raw_pos_d;
```

```
                int16_t paccuracy = currentPosition->pacc;
                int16_t nsat      = currentPosition->numsv;

               findDistance(refPstnECEF,curPstnECEF);

        }
}
/*********************************************************************
 *      This computes the distance between two ECEF position
 *              inputs: reference ECEF position of UAV,
 *                      current ECEF position of UAV
 *              output: none
 *********************************************************************/
void findDistance( const Vector<3> &position1, const Vector<3> &position2)
{
    Vector<3> deltaP ( position1 - position2 );

       double relDist        =     sqrt( deltaP[0]*deltaP[0]+
                                         deltaP[1]*deltaP[1]+
                                         deltaP[2]*deltaP[2] );
       if (meters)
                dist =  relDist;
       else if (feet)
                dist =  relDist * C_M2FT;
       else if (yards)
                dist =  relDist * 1.0/C_YD2M;
       else if (miles)
                dist =  relDist * 1.0/C_MI2M;
       else if (nautical)
                dist =  relDist * 1.0/C_NMI2M;
       else
                dist = relDist;

            Vector<3> position1LLH(ECEF2llh(position1));
            Vector<3> position2LLH(ECEF2llh(position2));

                      position1LLH[0] = position1LLH[0] * C_RAD2DEG;
                      position1LLH[1] = position1LLH[1] * C_RAD2DEG;
                      position1LLH[2] = position1LLH[2];

                      position2LLH[0] = position2LLH[0] * C_RAD2DEG;
                      position2LLH[1] = position2LLH[1] * C_RAD2DEG;
                      position2LLH[2] = position2LLH[2];

        alpha  =asin ( (position2LLH[2] - position1LLH[2])/dist)*C_RAD2DEG ;

            saveResults(position1LLH,position2LLH);
}
/*********************************************************************
 *      This function logs the distance and position of the UAV.
```

```
 *              inputs: reference ECEF position of UAV,
 *                      current ECEF position of UAV
 *              output: none
 ************************************************************************/
void saveResults(const Vector<3> &pstn1, const Vector<3> &pstn2)
{

 int i;
 struct timeval t;

 for( i = 0;i<=2;i++)
  {  if (i == 0)
      { gettimeofday(&t,0);

        fprintf(pstnLogFile,"\n%ld.%6ld %#10.4f %#17.4f %#14.4f %#12.2f\n",
                t.tv_sec,(t.tv_usec)%1000000,
                pstn1[i],pstn2[i],dist,alpha);
        printf("\n%ld.%6ld %#10.4f %#17.4f %#12.4f %#12.2f\n",
                t.tv_sec,(t.tv_usec)%1000000,
                pstn1[i],pstn2[i],dist,alpha);
      }
      else{
         printf(" %#27.4f %#17.4f\n",pstn1[i],pstn2[i]);
         fprintf(pstnLogFile," %27.4f %17.4f",pstn1[i],
                 pstn2[i]);
         fprintf(pstnLogFile,"\n");}
  }

}
/************************************************************************
 *      This function converts LLH(deg,min,sec) to decimal degrees.
 *              input: (deg,mim,sec)
 *              output: decinal degree
 ************************************************************************/
double llh2deg(char *argv)
{

        int n = strlen(argv);

        char sign = *(argv);
        char cardinalPoint = *(argv + (n -1));

        if ( (sign == '-'|| isdigit(sign)) && isdigit(cardinalPoint) )
                return atof(argv);

        else if (isdigit(sign) && isalpha(cardinalPoint))
             {

                char coord[n];
                int i = 0;
```

```
                    while(i <= (n-1))
                    {

                         coord[i] = (int)*(argv+i);
                            i++;
                            }
                double coordDeg = atoi(&coord[0])/10000 +
                 ((atoi(&coord[0])/100)%100)/60.0 +
                (atoi(&coord[0])%100)/3600.0;

          switch ( cardinalPoint ) {

             case 'N':
             case 'n':
                 return coordDeg;
             case 'S':
             case 's':
                 return -coordDeg;
             case 'E':
             case 'e':
                 return coordDeg;
             case 'W':
             case 'w':
                 return -coordDeg;
             default:
                { cout << "Unkown cardinal point " << endl;
                     exit(EXIT_FAILURE);
                     }
                 }
             }
        else
               { cout << " invalid coordinates!" << endl;
                     exit(EXIT_FAILURE); }
}

 /*************************************************************************
 *      This function initializes the log file.
 *              input: none
 *              output: none
 *************************************************************************/
void initLogFile()
{

 time_t         myt;
 struct tm *  ltime;
 char         datestr[80];
    if (( pstnLogFile = fopen (pstnlog,"w"))==NULL)
          {
            cout<< "ERROR: could not open file :"<< *pstnlog<<endl;
            exit(0);
          }
```

```
 time(&myt);
 ltime = localtime(&myt);
 strftime (datestr,80,"DATE: %d-%m-%Y",ltime);
 fprintf(pstnLogFile,
 "---------------------------------%s-------------------------------------\n\n"
                    "University of Dayton in collaboration with\n"
                    "Air Force Research Laboratory\n"
                    "Sensors Directorate\n",pstnlog);
 fputs(datestr,pstnLogFile);
 fprintf(pstnLogFile,"\n\nThis file contains the distance"
                    " and the angle from a reference position to \n"
                    "the UAV.  Position is in LLH(latitude longitude altitude)
                    (deg)(m) "
                    "coordinates\n and "
                    "Distance in (%s)."
                    "\n\n"
                    "%s %22s %17s %15s %12s \n",units(),
                    "Time","Ref Position","UAV Position","Distance","Angle(deg)");

 puts(datestr);
 printf("%s %22s %17s %15s %12s \n","Time","Ref Position",
   "UAV Position","Distance","Angle(deg)");
}

/*********************************************************************
 *      This function returns the unit of results.
 *              input: none
 *              output: unit(feet,meters,miles,nautical miles)
 *********************************************************************/
const char *units()
{
        if (meters)
                return me;
        else if (feet)
                return fe;
        else if (yards)
                return ya;
        else if (miles)
                return Mi;
        else if (nautical)
                return Na;
        else
                return me;
}

/*****************************
 Exit handler
*****************************/
void cleanup(int sigint)
{
     cout << endl
```

```
        << "EXITING PROGRAM! Data is saved in "<<pstnlog
        << endl << endl;

    fclose(pstnLogFile);
    exit(EXIT_SUCCESS);
}
```

7.2 Test Data

	Actual dist(ft)	GPS Measured distance (ft)										
	Test	1	2	3	4	5	6	7	8	9	Mean	Mean
	# Satellites	7	7	7	8	8	8	9	9	9		error
1	10	9.7	19.8	9.6	8.4	14.3	11.2	5.5	9.2	13.7	11.27	1.27
2	20	26.6	19.2	17.3	23.3	19.0	20.5	15.0	20.5	19.3	20.08	0.08
3	40	38.8	42.8	37.8	36.6	42.2	39.2	36.0	36.7	39.0	38.79	1.21
4	80	78.3	78.5	77.2	75.5	81.4	79.3	76.0	72.8	78.2	77.47	2.53
5	160	158.5	158.3	156.8	154.9	161.0	159.4	155.0	153.7	156.2	157.09	2.91
6	320	316.5	316.2	315.1	312.7	319.2	318.1	315.0	312.5	314.6	315.54	4.46
7	640	632.7	636.5	634.5	635.1	633.4	632.9	630.0	631.0	634.8	633.43	6.57
8	1280	1275.2	1277.8	1273.0	1278.3	1272.4	1276.1	1273.0	1270.2	1275.6	1274.62	5.38

Figure 7.2: Test results for eight positions for the position reference code.

CHAPTER 8

APPENDIX II
FORMATION CONTROL CODE

8.1 Real Time Simulation Code

8.1.1 Gazebo World File (Heli-Formation.world)

```
*******************************Heli-Formation.world*****************
<!-- ***************************************************************
* Declare global variables
*****************************************************************-->
<?xml version="1.0"?>
<gz:world xmlns:gz="http://playerstage.sourceforge.net/gazebo/xmlschema/#gz"
xmlns:model="http://playerstage.sourceforge.net/gazebo/xmlschema/#model"
xmlns:sensor="http://playerstage.sourceforge.net/gazebo/xmlschema/#sensor"
xmlns:window="http://playerstage.sourceforge.net/gazebo/xmlschema/#window"
xmlns:param="http://playerstage.sourceforge.net/gazebo/xmlschema/#params"
xmlns:ui="http://playerstage.sourceforge.net/gazebo/xmlschema/#params">
<param:Global>
<gravity>0.0 0.0 -9.8</gravity>
<stepTime>0.1</stepTime>
</param:Global>

<!-- ***************************************************************
* Define world environment
*****************************************************************-->
  <model:SkyDome>
    <toneMap>skyTones.jpg</toneMap>
    <timeLapse>true</timeLapse>
  </model:SkyDome>

  <model:LightSource>
    <id>light1</id>
    <xyz>0.000 10.000 100.000</xyz>
    <ambientColor>0.2, 0.2, 0.2</ambientColor>
    <diffuseColor>0.8, 0.8, 0.8</diffuseColor>
```

```
    <specularColor>0.2, 0.2, 0.2</specularColor>
    <attenuation>0.5, 0.0, 0.00</attenuation>
  </model:LightSource>

  <!--model:GroundPlane>
    <id>ground1</id>
    <color>0.0 0.5 0.0</color>
    <textureFile>grid.ppm</textureFile>
  </model:GroundPlane>-->

  <model:Terrain>
    <xyz>0.0 0.0 0.00</xyz>
    <color>0.0 0.5 0.0</color>
    <terrainFile>nati.gzb</terrainFile>
    <textureFile>grid.ppm</textureFile>
  </model:Terrain>

<!-- ********************************************************************
* Define UAV and GPS sensor models
*********************************************************************-->
<model:OsuHeli>
      <id>heli1</id>
    <xyz>144 160 30 </xyz>
    <!--xyz>157 148 30 </xyz-->
     <rpy>0 0 0</rpy>
     <updateRate>10</updateRate>
   <model:GarminGPS>
      <id>heli1_gps</id>
      <xyz>0.0 0.0 0.26</xyz>
      <rpy>0.0 0.0 0.0</rpy>
      <updateFreq>10</updateFreq>
    </model:GarminGPS>
</model:OsuHeli>

<!-- ********************************************************************
* Define the second UAV, GPS and camera models
*********************************************************************-->
<model:OsuHeli>
      <id>heli2</id>
     <xyz>154 140 30 </xyz>
    <!--xyz>144 150 30 </xyz-->
     <rpy>0 0 0</rpy>
    <updateRate>10</updateRate>
   <model:GarminGPS>
      <id>heli2_gps</id>
      <xyz>0.0 0.0 0.26</xyz>
      <rpy>0.0 0.0 0.0</rpy>
      <updateFreq>10</updateFreq>
    </model:GarminGPS>
   <model:ObserverCam>
    <id>userCam0</id>
```

```
    <xyz>-5.0 0.0 4.0</xyz>
    <rpy>0.0 20.0 0.0</rpy>
    <imageSize>700 580</imageSize>
    <updateRate>10</updateRate>
    <hfov>90.0</hfov>
    <nearClip>0.5</nearClip>
    <farClip>500.0</farClip>
    <rollLock>true</rollLock>
    <renderMethod>GLX</renderMethod>
    <saveFrames>true</saveFrames>
    <savePath>cam0_frames</savePath>
  </model:ObserverCam>
</model:OsuHeli>

<!-- ********************************************************************
* Define an independent observer camera.
*********************************************************************-->
<model:ObserverCam>
    <id>userCam1</id>
    <xyz>145 140 20</xyz>
    <rpy>0.0 -30.0 45</rpy>
    <imageSize>700 580</imageSize>
    <updateRate>10</updateRate>
    <hfov>90.0</hfov>
    <nearClip>0.5</nearClip>
    <farClip>500.0</farClip>
    <rollLock>true</rollLock>
    <renderMethod>GLX</renderMethod>
    <saveFrames>true</saveFrames>
    <savePath>cam1_frames</savePath>
  </model:ObserverCam>
*********************************************************************
```

8.1.2 Player Configuration Files

player-heli1.cfg

```
**************************player_heli1.cfg*********************
# Desc: Player configuration  file for Heli1
# Date: 16 May 2007

# Main simulation interface
driver
(
name "gz_sim"
provides ["simulation:0"]
)

# Robot position interface
driver
(
```

```
name "gz_position3d"
provides ["position3d:0"]
gz_id "heli1"
)

# GPS interface
driver
(
   name "gz_gps"
   provides ["gps:0"]
   gz_id "heli1_gps"
)
************************************************************
```

player-heli2.cfg

```
*****************************player_heli2.cfg****************
# Desc: Player configuration  file for Heli2
# Date: 16 May 2007

# Main simulation interface
driver
(
name "gz_sim"
provides ["simulation:0"]
)
# Robot position interface
driver
(
name "gz_position3d"
provides ["position3d:0"]
gz_id "heli2"
)
# GPS interface
driver
(
   name "gz_gps"
   provides ["gps:0"]
   gz_id "heli2_gps"
)
************************************************************
```

8.1.3 Control Code for UAV1 (heli1-test.cpp)

```
/************************************************************
* heli1_test.cpp
*
*   Formation control code.
* This code runs independently as process1 and controls
* the position of UAV1. It communicates with a Player
* interface which in turn interfaces with Gazebo robots.
```

```
*
* Data from the other processes obtained through shared memory.
*
* Author: Boakye Dankwa
* Date: May 23rd 2007
*************************************************************/
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/time.h>
#include <sys/types.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <math.h>
#include <playerclient.h>
#include <sys/msg.h>

#include <fstream>
#include <string>

#include <timer.h>
#include "Heli.h"
#include "cccs.h"
#include "Heliformation.h"


#define  BUFSIZE 256
//update time for control computation
#define  UPDATETIM 0.1

using namespace std;

/*****************************
Global variables
*****************************/
int shmh1_id;
int shmh2_id;
int shmh3_id;
int shmt_id;

int sem1_id;
int sem2_id;
int semt_id;

Heli1_data *hp1;
Heli2_data *hp2;
Heli3_data *hp3;
```

```
Target_data *tp;

PlayerClient *client;
Position3DProxy *pp;
GpsProxy *gp;
int do_shutdown = 0;

/****************************
Utility functions
****************************/
void Timer_Expire(int signal) {}
void cleanup();
void save(ofstream&,Vector<3>&, Vector<3>&, Vector<3>&,
Vector<3>&, Vector<3>&, Vector<6>&);
void Shutdown(int signal)
{
    do_shutdown = 1;
    cleanup();
 }

/*************************************************************
* Converts a (latitude,longitude) point to an (x,y)
* point in meters
*************************************************************/
static void ll2xyz( const double lat,
const double lon,
const double alt,
double &x,
double &y,
  double &z,
double A,
double B)
{
double R=6378.155*1000;
double R2=R*cos(A*pi/180);
x = (lon-B)/360*2*pi*R2;
y = (lat-A)/360*2*pi*R;
z = alt;
}

/*************************************************************
* Function prototypes for initializing the shared memory.
*************************************************************/
Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret);
Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret);
Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret);
Target_data *initialize_target_shared_memory(int key,int *shm_id_ret);

/*************************************************************
* Function prototypes for initializing the semaphores.
*************************************************************/
```

```
int initialize_heli1_sem(const char* key, int sem_id_ret);
int initialize_heli2_sem(const char* key, int sem_id_ret);
int initialize_target_sem(const char* key, int sem_id_ret);

/*************************************************************
* Function prototypes for shared mem. synchronization
*************************************************************/
void lock_heli1_sem_for_writing(int sem_id_ret);
void lock_heli2_sem(int sem_id_ret);
void lock_target_sem(int sem_id_ret);
void release_sem(int sem_id_ret);

int main(int argc, char **argv)
{


    int world_shm_id1;
     int world_shm_id2;
     int world_shm_id3;
     int world_shm_id4;

/*****************************************
* Naming scheme for the logfile.
* Include current name and time
* to name of file.
*****************************************/
time_t rawtime;
time(&rawtime);

string uavname = "helo1";
string  time_now = ctime(&rawtime);
string  extension = ".txt";

string  temp = uavname + time_now + extension;

char filename[35];
memset( filename, '\0', 35 );
temp.copy( filename, 35 );



ofstream outfile(filename,ios::app);

/************************Initalize share memory and semaphores************/


    hp1  = initialize_heli1pos_shared_memory(HELI1_POS_SHM_KEY,&world_shm_id1);
    hp2 = initialize_heli2pos_shared_memory(HELI2_POS_SHM_KEY,&world_shm_id2);
    hp3 = initialize_heli3pos_shared_memory(HELI3_POS_SHM_KEY,&world_shm_id3);
    tp  = initialize_target_shared_memory(TARGET_POS_SHM_KEY,&world_shm_id4);
```

```
    sem1_id = initialize_heli1_sem(HELI1_POS_SEM_KEY, SEM1_KEY);
    sem2_id = initialize_heli2_sem(HELI2_POS_SEM_KEY,SEM2_KEY);
    semt_id = initialize_target_sem(TARGET_POS_SEM_KEY,SEMT_KEY);
/*************************End Initalize share memory*********************/

/*******************************************
* Initialize formation parameters
* from the target shared memory.
*******************************************/

lock_target_sem(semt_id);

double  heli1_heli2  = tp->formation.fmtn;
double  altitude  = tp->formation.altitude;
int trajectory = tp->formation.trajectory;
double  radius = tp->formation.circle_radius;
double speed = tp->formation.target_speed;
double heading = tp->formation.line_heading;
double orientation = tp->formation.formation_orientation;
double slope = tp->formation.line_slope;
double X0 = tp->formation.start_north;
double Y0 = tp->formation.start_east;

release_sem(sem1_id);

double  a = 1.0000;
double  b  = 100.0000;
double  c  = (heli1_heli2)*(heli1_heli2)/log(b);

double delta_it =  heli1_heli2/2.0;

/*****************************************************
*
* heli1_helii2
*     10**********0**********02
*
*     aij  = 1.0;
*     bij = 100.0;
*     cij = delta_ij^2/log(aij*bij);
*****************************************************/

/************Done Initializing formation parameters*/


    X_def X;
    X_def X2;
    Xt_def Xt;
    Vector<3>Z;

    X_dot_def Xdot;
```

```
    Xt_dot_def Xtdot;
    Vector<3>Zdot;
    Vector<6>data;

    Vector<3>init_cond(0.0,0.0,0.0);


    X.Vel  = init_cond;
    Z = init_cond;
    Zdot = init_cond;
    Xdot.Pos_dot = init_cond;

    double a12 = a;
    double b12 = b;
    double c12 = c;

/*****************************
* Origin in Gazebo world
*****************************/
    double x = 0.0;
    double y = 0.0;
    double z = 0.0;
    double ref1 = 0.000000;
    double ref2 = 178.511256;

/*****************************
* variables Initializatoin
*****************************/
    Vector<3>aij(0.0,a12,0.0);
    Vector<3>bij(0.0,b12,0.0);
    Vector<3>cij(0.0,c12,0.0);


    Vector<3>X_position(0.0,0.0,0.0);
    Vector<3>X2_position(0.0,0.0,0.0);
    Vector<3>X3_position(0.0,0.0,0.0);
    Vector<3>Xt_position(0.0,0.0,0.0);

    Vector<3>new_X_position(0.0,0.0,0.0);
    Vector<3>new_X_velocity(0.0,0.0,0.0); //uvw
    //pqr
    Vector<3>   X_velocity(0.0,0.0,0.0);
    Vector<3>new_Xt_position(0.0,0.0,0.0);



/*****************************
* step time
*****************************/
    double t = 0.0;
```

```
    double dt = 0.1;

/****************************
* timer used during
* development.
****************************/
    unsigned long elapsed_time = 0;

   struct itimerval interval;

   stopwatch_t  T;

/****************************
* Register signal handlers
****************************/
    signal(SIGINT, Shutdown);       // Handles <ctrl-c> command from user
    signal(SIGTERM, Shutdown);
    signal(SIGKILL, Shutdown);
    signal(SIGALRM, Timer_Expire);   // Handles timer expiration



/*************************************************
* Create all modules, messages, and connections
* NOTE: ONLY THIS PART OF THE CODE
* CHANGES FOR IMPLEMENTATION ON THE
* REAL ROBOTS.
*************************************************/
    client = new PlayerClient();
/***********************************
* connect to player interface
***********************************/
    if (client->Connect("localhost", 7000/*PLAYER_PORTNUM*/))
 //if (client->Connect("192.168.1.3",7000 /*hp1->network.Port*/))
    {
        printf("Could not connect to Player\n");
        exit(EXIT_FAILURE);
    }

   client->SetDataMode(PLAYER_DATAMODE_PULL_NEW);

    pp = new Position3DProxy(client, 0, 'a');
    gp = new GpsProxy(client,0,'r');

    pp->SelectPositionMode(0);
    //pp->SetOdometry(x,y,z,theta,phi,psi);
    pp->SetMotorState(1);
/*********************End player interface**********************/

/*****************************************
*  Attempt to set the timer to UPDATE Hz
```

```
*****************************************/
    interval.it_value.tv_sec = 0;
    interval.it_value.tv_usec = (long)(UPDATETIM*1e6);
    interval.it_interval.tv_sec = 0;
    interval.it_interval.tv_usec = (long)(UPDATETIM*1e6);
    if ( setitimer(ITIMER_REAL, &interval, NULL) )
    {
        printf("Could not set the timer\n");
        exit(EXIT_FAILURE);
    }

 int n = 1;
/*****************************************
* Main control loop
*****************************************/
 while (!do_shutdown)
    {
start(&T);

if ((client->Read()) < 0 )
{ printf("\ncould not read from PlayerClient object");
exit(EXIT_FAILURE);
}

/***********************************
* Get current position of UAV
***********************************/
        ll2xyz( gp->latitude,
gp->longitude,
gp->altitude,
x,
y,
z,
ref1,
ref2);

X_position[0] = x;
        X_position[1] = y;
        X_position[2] = z;

X_velocity[0] = pp->XSpeed();
X_velocity[1] = pp->YSpeed();
X_velocity[2] = pp->ZSpeed();

/*******************************
* Lock resources and update
* with current position data.
*******************************/
lock_heli1_sem_for_writing(sem1_id);

        hp1->position.x = X_position[0];
```

```
        hp1->position.y = X_position[1];
        hp1->position.z = X_position[2];

release_sem(sem1_id);
/************End Updating Virtual memory for UAV1****/

/****************************************
* Lock resources and acquire
* current position of UAV2.
****************************************/
        lock_heli2_sem(sem2_id);

X2_position[0] = hp2->position.x;
        X2_position[1] = hp2->position.y;
        X2_position[2] = hp2->position.z;

release_sem(sem2_id);
/************End acquiring position of UAV2***/
/****************************************
* Lock resources and acquire
* current position of target.
****************************************/

lock_target_sem(semt_id);

Xt_position[0] = tp->position.x;
        Xt_position[1] = tp->position.y;
        Xt_position[2] = tp->position.z;

        release_sem(semt_id);
/************End acquiring position of target***/
/****************************************
* Make sure we're doing calculations
in 2D.
****************************************/

X_position[2]  = 0.0;
X2_position[2]  = 0.0;
Xt_position[2] = 0.0;


X.Pos = X_position;

X2.Pos = X2_position;

Xt.Pos = Xt_position;

/****************************************
* Computes the next state using Rk4.
****************************************/
step( &Xdot,
```

```
&Xtdot,
Zdot,
&X,
&Xt,
Z,
&X2,
aij,
bij,
cij,
data,
delta_it,
trajectory,
radius,
speed,
heading,
orientation,
slope,
X0,
Y0,
t,
dt);

t = t + dt;


new_X_position = X.Pos;

new_X_velocity = Xdot.Pos_dot;

/**************Sending these controllers to the robot**************/
/*****************************************
* NOTE: we send the control every 0.5 seconds because
* the fastest posible rate allowable by the real
* UAV is 0.25 seconds.
*****************************************************/
if (n%5 == 0 ){
pp->SetSpeed( new_X_position[0],//x
new_X_position[1],//y
altitude,
0.0,
0.0,
0.0);
n = 0;
save(outfile, X_position, new_X_position, X_velocity,
 new_X_velocity, Xt_position, data);
}
/***************End Sending*****************************************/



       // Wait for timer to expire...
```

```
select(0, NULL, NULL, NULL, NULL);
n++;

elapsed_time = stop(&T);

printf("\nelapsed time:  %ld usec", elapsed_time);


    }

    // Stop the timer
    interval.it_value.tv_sec = 0;
    interval.it_value.tv_usec = 0;
    interval.it_interval.tv_sec = 0;
    interval.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &interval, NULL);


    outfile.close();

    return 0;

}

/****************************************
* Function definitions
****************************************/
Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli1_data *h1;

  printf("Attempt to get shared memory of heli 1 Position.\n");
  flag = IPC_CREAT | 0777;
  printf("%d",sizeof(Heli1_data));
  shmh1_id = shmget(key,sizeof(Heli1_data),flag);
  if(shmh1_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got shared memory of vehicle 1 Position\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli1.Position\n");
  h1 = (Heli1_data *)shmat(shmh1_id,0,0);

  if((int)h1 == -1)
    {
      perror("error: shmat\n");
      exit(-1);
```

```
    }
  printf("Shared memory of heli1 Position attached.\n");
  *shm_id_ret=shmh1_id;

  printf("Shared memory of h1 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h1->x, h1->y, h1->z);
  return(h1);

}

Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli2_data *h2;

  printf("Attempt to get Heli2 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh2_id = shmget(key,sizeof(Heli2_data),flag);
  if(shmh2_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli2 Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of Heli2.\n");
  h2 = (Heli2_data *)shmat(shmh2_id,0,0);

  if((int)h2== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli2 attached.\n");
  *shm_id_ret=shmh2_id;

  printf("Shared memory of h2 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h2->x, h2->y, h2->z);
 return(h2);

}

Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli3_data *h3;

  printf("Attempt to get Heli3 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh3_id = shmget(key,sizeof(Heli3_data),flag);
```

```
  if(shmh3_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli3 Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli3.\n");
  h3 = (Heli3_data *)shmat(shmh3_id,0,0);

  if((int)h3== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of h3 attached.\n");
  *shm_id_ret=shmh3_id;

  printf("Shared memory of h3 initialized.\n");
 // printf("x = %f, y = %f, z = %f.\n",h3->x, h3->y, h3->z);
 return(h3);

}
Target_data *initialize_target_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Target_data *t1;

  printf("Attempt to get Target shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmt_id = shmget(key,sizeof(Target_data),flag);
  if(shmt_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Taget Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of vehicle2.\n");
  t1 = (Target_data *)shmat(shmt_id,0,0);

  if((int)t1== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of target attached.\n");
  *shm_id_ret=shmt_id;
```

```
    printf("Shared memory of target initialized.\n");
   // printf("x = %f, y = %f, z = %f.\n",t1->x, t1->y, t1->z);
 return(t1);

}

int initialize_heli1_sem(const char *key, int sem_id_ret  )
{
  int flag;
  int semt_id;
  struct sembuf;

   key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }

  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: sEmget\n");
      exit(-1);
    }

 return(semt_id);
}

int initialize_heli2_sem(const char* key, int sem_id_ret)
{
  int flag;
  int semt_id;
  struct sembuf;

  key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }

  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: sEmget\n");
      exit(-1);
```

```
    }

 return(semt_id);
}

int initialize_target_sem(const char* key, int sem_id_ret)
{
  int flag;
  int semt_id;
  struct sembuf;

   key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }

  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: sEmget\n");
      exit(-1);
    }

 return(semt_id);
}

void lock_heli1_sem_for_writing(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op  = 0;
 operations[0].sem_flg = 0;

 operations[1].sem_num  = 0;
 operations[1].sem_op  = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);
```

```
}

void lock_heli2_sem(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op  = 0;
 operations[0].sem_flg = 0;

 operations[1].sem_num  = 0;
 operations[1].sem_op  = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);


}

void lock_target_sem(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op  = 0;
 operations[0].sem_flg = 0;

 operations[1].sem_num  = 0;
 operations[1].sem_op  = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);


}

void release_sem(int semt_id)
{
  int rc;
```

```
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op  = -1;
 operations[0].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,1,&timeout);


}

void cleanup()
{
shmdt(hp1);
shmdt(hp2);
shmdt(hp3);
shmdt(tp);

shmctl(shmh1_id,IPC_RMID,0);
        shmctl(shmh2_id,IPC_RMID,0);
shmctl(shmh3_id,IPC_RMID,0);
shmctl(shmt_id,IPC_RMID,0);

semctl(sem1_id, 1, IPC_RMID);
semctl(sem2_id, 1, IPC_RMID);
semctl(semt_id, 1, IPC_RMID);

delete client;
        delete pp;
delete gp;

}

void save(ofstream& outfile, Vector<3>& P1, Vector<3>&P2,
            Vector<3>&V1, Vector<3>&V2, Vector<3>& T, Vector<6>& D)
{

time_t curr_time;
time(&curr_time);

struct tm  time_now;

localtime_r(&curr_time, &time_now);

outfile << time_now.tm_hour<<":" << time_now.tm_min <<":"
  << time_now.tm_sec<<"\t"
<< P1[0] <<"\t" << P1[1] <<"\t" << P1[2] <<"\t"
```

```
        << P2[0] <<"\t" << P2[1] <<"\t" << P2[2] <<"\t"
<< V1[0] <<"\t" << V1[1] <<"\t" << V1[2] <<"\t"
        << V2[0] <<"\t" << V2[1] <<"\t" << V2[2] <<"\t"
<< T[0] <<"\t" << T[1] <<"\t" << T[2] <<"\t"
<< D[0] <<"\t" << D[1] <<"\t" << D[2] <<"\t"<< D[3] <<"\t"
<< D[4] <<"\t" << D[5] <<"\t"
<< norm(P1,P2)<<"\n";

outfile.flush();
}
```

8.1.4 Control Code for UAV2 (heli2-test.cpp)

```
/*************************************************************
* heli2_test.cpp
*
*   Formation control code.
* This code runs independently as process2 and controls
* the position of UAV2. It communicates with a Player
* interface which in turn interfaces with Gazebo robots.
*
* Data from the other processes obtained through shared memory.
*
* Author: Boakye Dankwa
* Date: May 23rd 2007
*************************************************************/
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/time.h>
#include <sys/types.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <math.h>
#include <playerclient.h>
#include <sys/msg.h>

#include <string>
#include <fstream>

#include <timer.h>
#include "Heli.h"
#include "cccs.h"
#include "Heliformation.h"


#define  BUFSIZE 256
```

```
//update time from gazebo
#define  UPDATETIM 0.1

using namespace std;

/*****************************
Global variables
*****************************/
int shmh1_id;
int shmh2_id;
int shmh3_id;
int shmt_id;

int sem1_id;
int sem2_id;
int semt_id;

Heli1_data *hp1;
Heli2_data *hp2;
Heli3_data *hp3;
Target_data *tp;

PlayerClient *client;
Position3DProxy *pp;
GpsProxy *gp;
int do_shutdown = 0;

/*****************************
Utility functions
*****************************/
void Timer_Expire(int signal) {}
void cleanup();
void save(ofstream&,Vector<3>&, Vector<3>&, Vector<3>&,
Vector<3>&, Vector<3>&, Vector<6>&);
void Shutdown(int signal)
{
    do_shutdown = 1;
    cleanup();
 }

/*************************************************************
* Converts a (latitude,longitude) point to an (x,y)
* point in meters
/*************************************************************/
static void ll2xyz( const double lat,
const double lon,
const double alt,
double &x,
double &y,
  double &z,
double A,
```

```
double B)
{
double R=6378.155*1000;
double R2=R*cos(A*pi/180);
x = (lon-B)/360*2*pi*R2;
y = (lat-A)/360*2*pi*R;
z = alt;
}
/*************************************************************
* Function prototypes for initializing the shared memory.
*************************************************************/

Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret);
Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret);
Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret);
Target_data *initialize_target_shared_memory(int key,int *shm_id_ret);

/*************************************************************
* Function prototypes for initializing the semaphores.
*************************************************************/
int initialize_heli1_sem(const char* key, int sem_id_ret);
int initialize_heli2_sem(const char* key, int sem_id_ret);
int initialize_target_sem(const char* key, int sem_id_ret);

/*************************************************************
* Function prototypes for shared mem. synchronization
*************************************************************/
void lock_heli2_sem_for_writing(int sem_id_ret);
void lock_heli1_sem(int sem_id_ret);
void lock_target_sem(int sem_id_ret);
void release_sem(int sem_id_ret);

int main(int argc, char **argv)
{

int world_shm_id1;
     int world_shm_id2;
     int world_shm_id3;
     int world_shm_id4;

/*****************************************
* Naming scheme for the logfile.
* Include current name and time
* to name of file.
*****************************************/
time_t rawtime;
time(&rawtime);

string uavname = "helo2";
string  time_now = ctime(&rawtime);
string  extension = ".txt";
```

```
string  temp = uavname + time_now + extension;

char filename[35];
memset( filename, '\0', 35 );
temp.copy( filename, 35 );



ofstream outfile(filename,ios::app);

/*************************Initalize share memory**********************/

    hp1  = initialize_heli1pos_shared_memory(HELI1_POS_SHM_KEY,&world_shm_id1);
    hp2 = initialize_heli2pos_shared_memory(HELI2_POS_SHM_KEY,&world_shm_id2);
    hp3 = initialize_heli3pos_shared_memory(HELI3_POS_SHM_KEY,&world_shm_id3);
    tp  = initialize_target_shared_memory(TARGET_POS_SHM_KEY,&world_shm_id4);

    sem1_id = initialize_heli1_sem(HELI1_POS_SEM_KEY, SEM1_KEY);
    sem2_id = initialize_heli2_sem(HELI2_POS_SEM_KEY, SEM2_KEY);
    semt_id = initialize_target_sem(TARGET_POS_SEM_KEY, SEMT_KEY);
/*************************End Initalize share memory**********************/
/*******************************************
* Initialize formation parameters
* from the target shared memory.
*******************************************/

lock_target_sem(semt_id);
double  heli1_heli2  = tp->formation.fmtn;
double  altitude  = tp->formation.altitude;
int trajectory = tp->formation.trajectory;
double  radius = tp->formation.circle_radius;
double speed = tp->formation.target_speed;
double heading = tp->formation.line_heading;
double  orientation = tp->formation.formation_orientation;
double slope = tp->formation.line_slope;
double X0 = tp->formation.start_north;
double Y0 = tp->formation.start_east;

release_sem(sem1_id);

double  a = 1.0000;
double  b  = 100.0000;
double  c  = (heli1_heli2)*(heli1_heli2)/log(100.0);

double delta_it =  heli1_heli2/2.0;


/*************************************************
*
```

```
* heli1_helii2
*     10**********0**********02
*
*     aij  = 1.0;
*     bij = 100.0;
*     cij = delta_ij^2/log(aij*bij);
*************************************************/

/************Done Initializing formation parameters*/

    X_def X;
    X_def X1;
    Xt_def Xt;
    Vector<3>Z;

    X_dot_def Xdot;
    Xt_dot_def Xtdot;
    Vector<3>Zdot;
    Vector<6>data;


    Vector<3>init_cond(0.0,0.0,0.0);


    X.Vel  = init_cond;
    Z = init_cond;
    Zdot = init_cond;
    Xdot.Pos_dot = init_cond;



    double a21 = a;
    double b21 = b;
    double c21 = c;

/*****************************
* Origin in Gazebo world
*****************************/
    double x = 0.0;
    double y = 0.0;
    double z = 0.0;
    double ref1 = 0.000000;
    double ref2 = 178.511256;

/*****************************
* variables Initializatoin
*****************************/
    Vector<3>aij(0.0,a21,0.0);
    Vector<3>bij(0.0,b21,0.0);
    Vector<3>cij(0.0,c21,0.0);
```

```
    Vector<3>X_position(0.0,0.0,0.0);
    Vector<3>X1_position(0.0,0.0,0.0);
    Vector<3>X3_position(0.0,0.0,0.0);
    Vector<3>Xt_position(0.0,0.0,0.0);

    Vector<3>new_X_position(0.0,0.0,0.0);
    Vector<3>new_X_velocity(0.0,0.0,0.0);

    Vector<3>   X_velocity(0.0,0.0,0.0);
    Vector<3>new_Xt_position(0.0,0.0,0.0);

/*****************************
* step time
*****************************/
    double t = 0.0;
    double dt = 0.1;

/*****************************
* timer used during
* development.
*****************************/
    unsigned long elapsed_time = 0;

   struct itimerval interval;

   stopwatch_t  T;
/*****************************
* Signal handlers
*****************************/
    signal(SIGINT, Shutdown);       // Handles <ctrl-c> command from user
    signal(SIGTERM, Shutdown);
    signal(SIGKILL, Shutdown);
    signal(SIGALRM, Timer_Expire);   // Handles timer expiration


/*************************************************
* Create all modules, messages, and connections
* NOTE: ONLY THIS PART OF THE CODE
* CHANGES FOR IMPLEMENTATION ON THE
* REAL ROBOTS.
*************************************************/
    client = new PlayerClient();
/***********************************
* connect to player interface
***********************************/
    if (client->Connect("localhost", 7001/*PLAYER_PORTNUM*/))
 //if (client->Connect("192.168.1.3",7001 /*hp2->network.Port*/))
    {
        printf("Could not connect to Player\n");
        exit(EXIT_FAILURE);
```

```
    }

   client->SetDataMode(PLAYER_DATAMODE_PULL_NEW);

    pp = new Position3DProxy(client, 0, 'a');
    gp = new GpsProxy(client,0,'r');

    pp->SelectPositionMode(0);
    //pp->SetOdometry(x,y,z,theta,phi,psi);
    pp->SetMotorState(1);
   /*********************End player interface*********************/

/*******************************************
*  Attempt to set the timer to UPDATE Hz
*******************************************/
    interval.it_value.tv_sec = 0;
    interval.it_value.tv_usec = (long)(UPDATETIM*1e6);
    interval.it_interval.tv_sec = 0;
    interval.it_interval.tv_usec = (long)(UPDATETIM*1e6);
    if ( setitimer(ITIMER_REAL, &interval, NULL) )
    {
        printf("Could not set the timer\n");
        exit(EXIT_FAILURE);
    }

 int n = 1;
/*******************************************
* Main control loop
*******************************************/
 while (!do_shutdown)
    {
start(&T);

if ((client->Read()) < 0 )
{ printf("\ncould not read from PlayerClient object");
exit(EXIT_FAILURE);
}

/***********************************
* Get current position of UAV
***********************************/
       ll2xyz( gp->latitude,
gp->longitude,
gp->altitude,
x,
y,
z,
ref1,
ref2);

X_position[0] = x;
```

```
        X_position[1] = y;
        X_position[2] = z;

        X_velocity[0] = pp->XSpeed();
X_velocity[1] = pp->YSpeed();
X_velocity[2] = pp->ZSpeed();

/*******************************
* Lock resources and update
* with current position data.
*******************************/

lock_heli2_sem_for_writing(sem2_id);

        hp2->position.x = X_position[0];
        hp2->position.y = X_position[1];
        hp2->position.z = X_position[2];

release_sem(sem2_id);

/************End Updating Virtual memory for UAV1****/

/****************************************
* Lock resources and acquire
* current position of UAV2.
****************************************/
        lock_heli1_sem(sem1_id);

X1_position[0] = hp1->position.x;
        X1_position[1] = hp1->position.y;
        X1_position[2] = hp1->position.z;

        release_sem(sem1_id);

/************End acquiring position of UAV2***/
/****************************************
* Lock resources and acquire
* current position of target.
****************************************/

    lock_target_sem(semt_id);

Xt_position[0] = tp->position.x;
        Xt_position[1] = tp->position.y;
        Xt_position[2] = tp->position.z;

release_sem(semt_id);
/************End acquiring position of target***/
/****************************************
* Make sure we're doing calculations
in 2D.
```

```
****************************************/

X_position[2]  = 0.0;
X1_position[2] = 0.0;
Xt_position[2] = 0.0;

X.Pos = X_position;

X1.Pos = X1_position;

Xt.Pos = Xt_position;
/****************************************
* Compute next position of UAV1.
****************************************/
step( &Xdot,
&Xtdot,
Zdot,
&X,
&Xt,
Z,
&X1,
aij,
bij,
cij,
data,
delta_it,
trajectory,
radius,
speed,
heading,
orientation,
slope,
X0,
Y0,
t,
dt);

t = t + dt;


new_X_position = X.Pos;

new_X_velocity = Xdot.Pos_dot;

/**************Sending these controllers to the robot**************/
/****************************************
* NOTE: we send the control every 0.5 seconds because
* the fastest posible rate is 0.25 seconds.
****************************************************************/
if (n%5 == 0 ){
pp->SetSpeed( new_X_position[0],//x
```

```
new_X_position[1],//y
altitude,
0.0,
0.0,
0.0);
n = 0;
save(outfile, X_position, new_X_position, X_velocity,
 new_X_velocity, Xt_position, data);
}
/***************End Sending*****************************************/



       // Wait for timer to expire...
select(0, NULL, NULL, NULL, NULL);
n++;

elapsed_time = stop(&T);

printf("\nelapsed time:  %ld usec", elapsed_time);


    }

    // Stop the timer
    interval.it_value.tv_sec = 0;
    interval.it_value.tv_usec = 0;
    interval.it_interval.tv_sec = 0;
    interval.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &interval, NULL);

    outfile.close();

    return 0;

}

/****************************************
* Function definitions
****************************************/

Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli1_data *h1;

  printf("Attempt to get shared memory of heli 1 Position.\n");
  flag = IPC_CREAT | 0777;
  printf("%d",sizeof(Heli1_data));
  shmh1_id = shmget(key,sizeof(Heli1_data),flag);
  if(shmh1_id<0)
```

```
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got shared memory of vehicle 1 Position\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli1.Position\n");
  h1 = (Heli1_data *)shmat(shmh1_id,0,0);

  if((int)h1 == -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli1 Position attached.\n");
  *shm_id_ret=shmh1_id;


  printf("Shared memory of h1 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h1->x, h1->y, h1->z);
  return(h1);

}

Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli2_data *h2;

  printf("Attempt to get Heli2 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh2_id = shmget(key,sizeof(Heli2_data),flag);
  if(shmh2_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli2 Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of Heli2.\n");
  h2 = (Heli2_data *)shmat(shmh2_id,0,0);

  if((int)h2== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli2 attached.\n");
  *shm_id_ret=shmh2_id;
```

```
  printf("Shared memory of h2 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h2->x, h2->y, h2->z);
 return(h2);

}

Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli3_data *h3;

  printf("Attempt to get Heli3 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh3_id = shmget(key,sizeof(Heli3_data),flag);
  if(shmh3_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli3 Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli3.\n");
  h3 = (Heli3_data *)shmat(shmh3_id,0,0);

  if((int)h3== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of h3 attached.\n");
  *shm_id_ret=shmh3_id;

  printf("Shared memory of h3 initialized.\n");
 // printf("x = %f, y = %f, z = %f.\n",h3->x, h3->y, h3->z);
 return(h3);

}
Target_data *initialize_target_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Target_data *t1;

  printf("Attempt to get Target shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmt_id = shmget(key,sizeof(Target_data),flag);
  if(shmt_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
```

```
    }
  printf("Got Taget Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of vehicle2.\n");
  t1 = (Target_data *)shmat(shmt_id,0,0);

  if((int)t1== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of target attached.\n");
  *shm_id_ret=shmt_id;

    printf("Shared memory of target initialized.\n");
   // printf("x = %f, y = %f, z = %f.\n",t1->x, t1->y, t1->z);
 return(t1);

}


int initialize_heli1_sem(const char* key, int sem_id_ret)
{
  int flag;
  int semt_id;
  struct sembuf;

   key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }

  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: sEmget\n");
      exit(-1);
    }

 return(semt_id);
}

int initialize_heli2_sem(const char* key, int sem_id_ret)
{
  int flag;
  int semt_id;
```

```
  struct sembuf;

   key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }
  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: sEmget\n");
      exit(-1);
    }

 return(semt_id);
}

int initialize_target_sem(const char* key, int sem_id_ret)
{
  int flag;
  int semt_id;
  struct sembuf;

  key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }

  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: sEmget\n");
      exit(-1);
    }

 return(semt_id);
}

void lock_heli2_sem_for_writing(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;
```

```
 operations[0].sem_num  = 0;
 operations[0].sem_op   = 0;
 operations[0].sem_flg = 0;

 operations[1].sem_num  = 0;
 operations[1].sem_op   = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);


}

void lock_heli1_sem(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op   = 0;
 operations[0].sem_flg = 0;

 operations[1].sem_num  = 0;
 operations[1].sem_op   = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);


}

void lock_target_sem(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op   = 0;
 operations[0].sem_flg = 0;
```

```
 operations[1].sem_num  = 0;
 operations[1].sem_op   = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);


}

void release_sem(int semt_id)
{
  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op   = -1;
 operations[0].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,1,&timeout);


}
void cleanup()
{
shmdt(hp1);
shmdt(hp2);
shmdt(hp3);
shmdt(tp);

shmctl(shmh1_id,IPC_RMID,0);
        shmctl(shmh2_id,IPC_RMID,0);
shmctl(shmh3_id,IPC_RMID,0);
shmctl(shmt_id,IPC_RMID,0);

semctl(sem1_id, 1, IPC_RMID);
semctl(sem2_id, 1, IPC_RMID);
semctl(semt_id, 1, IPC_RMID);

delete client;
        delete pp;
delete gp;

}
```

```
void save(ofstream& outfile, Vector<3>& P1, Vector<3>&P2,
 Vector<3>&V1, Vector<3>&V2, Vector<3>& T, Vector<6>& D)
{

time_t curr_time;
time(&curr_time);

struct tm  time_now;

localtime_r(&curr_time, &time_now);

outfile << time_now.tm_hour<<":" << time_now.tm_min <<":"
  << time_now.tm_sec<<"\t"
<< P1[0] <<"\t" << P1[1] <<"\t" << P1[2] <<"\t"
        << P2[0] <<"\t" << P2[1] <<"\t" << P2[2] <<"\t"
<< V1[0] <<"\t" << V1[1] <<"\t" << V1[2] <<"\t"
        << V2[0] <<"\t" << V2[1] <<"\t" << V2[2] <<"\t"
<< T[0] <<"\t" << T[1] <<"\t" << T[2] <<"\t"
<< D[0] <<"\t" << D[1] <<"\t" << D[2] <<"\t"<< D[3] <<"\t"
<< D[4] <<"\t" << D[5] <<"\t"
<< norm(P1,P2)<<"\n";

outfile.flush();
}
```

8.1.5 Code for the Virtual Target (target-test.cpp)

```
/*************************************************************
* target_test.cpp
*
*   This program runs as the virtual target (process3)
*     and does not interface with Player or Gazebo.
*
* Author: Boakye Dankwa
* Date: May 23rd 2007
*************************************************************/
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>
#include <sys/time.h>
#include <sys/types.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <math.h>
#include <sys/msg.h>
```

```
#include <timer.h>
#include "Heli.h"
#include "cccs.h"
#include "Heliformation.h"

#define  BUFSIZE 256
//update time for target
#define  UPDATETIM 0.5

/*****************************
Global variables
*****************************/
int shmh1_id;
int shmh2_id;
int shmh3_id;
int shmt_id;

int semt_id;

Heli1_data *hp1;
Heli2_data *hp2;
Heli3_data *hp3;
Target_data *tp;

/*****************************
Utility functions
*****************************/
int do_shutdown = 0;
void Timer_Expire(int signal) {}
void cleanup();
void Shutdown(int signal)
{
    do_shutdown = 1;
    cleanup();
 }


static void ll2xyz( const double lat,
const double lon,
const double alt,
double &x,
double &y,
  double &z,
double A,
double B)
{
double R=6378.155*1000;
double R2=R*cos(A*pi/180);
x = (lon-B)/360*2*pi*R2;
y = (lat-A)/360*2*pi*R;
z = alt;
```

```
}
/*************************************************************
* Function prototypes for initializing the shared memory.
*************************************************************/
Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret);
Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret);
Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret);
Target_data *initialize_target_shared_memory(int key,int *shm_id_ret);

/*************************************************************
* Function prototypes for initializing the semaphores.
*************************************************************/
int initialize_target_sem(const char* key);
void lock_target_sem(int sem_id_ret);
void release_sem(int sem_id_ret);



int main(int argc, char **argv)
{


     int world_shm_id1;
     int world_shm_id2;
     int world_shm_id3;
     int world_shm_id4;



/*************************Initalize share memory**********************/


    hp1  = initialize_heli1pos_shared_memory(HELI1_POS_SHM_KEY,&world_shm_id1);
    hp2 = initialize_heli2pos_shared_memory(HELI2_POS_SHM_KEY,&world_shm_id2);
    hp3 = initialize_heli3pos_shared_memory(HELI3_POS_SHM_KEY,&world_shm_id3);
    tp  = initialize_target_shared_memory(TARGET_POS_SHM_KEY,&world_shm_id4);


    semt_id = initialize_target_sem(TARGET_POS_SEM_KEY);
/*************************End Initalize share memory**********************/
/*******************************************
* Initialize formation parameters
* from the target shared memory.
*******************************************/
double  heli1_heli2  = tp->formation.fmtn;
double  altitude  = tp->formation.altitude;
int trajectory = tp->formation.trajectory;
double  radius = tp->formation.circle_radius;
double speed = tp->formation.target_speed;
double heading = tp->formation.line_heading;
double orientation = tp->formation.formation_orientation;
```

```
double slope = tp->formation.line_slope;
double X0 = tp->formation.start_north;
double Y0 = tp->formation.start_east;

double  a = 1.0000;
double  b  = 100.0000;
double  c  = (heli1_heli2)*(heli1_heli2)/log(100.0);

double delta_it =  heli1_heli2/2.0;

X_def X;
X_def X2;
X_def X3;
Xt_def Xt;
Vector<3>Z;

X_dot_def Xdot;
Xt_dot_def Xtdot;
Vector<3>Zdot;

Vector<6>data;


    Vector<3>Xt_position(0.0,0.0,0.0);
    Vector<3>Xt_orientation(0.0,0.0,0.0);

    Vector<3>new_Xt_position(0.0,0.0,0.0);
    Vector<3>new_Xt_orientation(0.0,0.0,0.0);

    double a22 = a;
    double a23 = a;
    double b22 = b;
    double b23 = b;
    double c22 = c;
    double c23 = c;


    Vector<3>aij(0.0,a22,a23);
    Vector<3>bij(0.0,b22,b23);
    Vector<3>cij(0.0,c22,c23);

/*****************************
* step time
*****************************/
    double t = 0.0;
    double dt = 0.1;

/*****************************
* timer used during
* development.
*****************************/
```

```
    unsigned long elapsed_time = 0;


   struct itimerval interval;
   stopwatch_t T;

/****************************
* Register signal handlers
****************************/
    signal(SIGINT, Shutdown);       // Handles <ctrl-c> command from user
    signal(SIGTERM, Shutdown);
    signal(SIGKILL, Shutdown);
    signal(SIGALRM, Timer_Expire);   // Handles timer expiration



    // Attempt to set the timer to UPDATE Hz
    interval.it_value.tv_sec = 0;
    interval.it_value.tv_usec = (long)(UPDATETIM*1e6);
    interval.it_interval.tv_sec = 0;
    interval.it_interval.tv_usec = (long)(UPDATETIM*1e6);
    if ( setitimer(ITIMER_REAL, &interval, NULL) )
    {
        printf("Could not set the timer\n");
        exit(EXIT_FAILURE);
    }


 int n = 0;
 while (!do_shutdown)//
    {

start(&T);

/*******************************
* Lock resources and update
* with current position data.
*******************************/
lock_target_sem(semt_id);

Xt_position[0] = tp->position.x;
        Xt_position[1] = tp->position.y;
        Xt_position[2] = tp->position.z;

release_sem(semt_id);
/************End Updating Virtual memory for target****/
Xt_position[2] = 0.0;


Xt.Pos = Xt_position;
/*****************************************
```

```
* Compute next target position.
****************************************/
step( &Xdot,
&Xtdot,
Zdot,
&X,
&Xt,
Z,
&X3,
aij,
bij,
cij,
data,
delta_it,
trajectory,
radius,
speed,
heading,
orientation,
slope,
X0,
Y0,
t,
dt);


t = t + dt;



new_Xt_position = Xt.Pos;

        usleep(1000);

 /*******************************
* Lock resources and update
* with current position data.
*******************************/
lock_target_sem(semt_id);

        tp->position.x = new_Xt_position[0];
        tp->position.y = new_Xt_position[1];
        tp->position.z = new_Xt_position[2];

release_sem(semt_id);
/************End Updating Virtual memory for target****/

       // Wait for timer to expire...
select(0, NULL, NULL, NULL, NULL);
n++;
```

```
elapsed_time = stop(&T);
printf("\nelapsed time: %ld usec", elapsed_time);

    }

    // Stop the timer
    interval.it_value.tv_sec = 0;
    interval.it_value.tv_usec = 0;
    interval.it_interval.tv_sec = 0;
    interval.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &interval, NULL);

return 0;

}
/****************************************
* Function definitions
****************************************/
Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli1_data *h1;

  printf("Attempt to get shared memory of heli 1 Position.\n");
  flag = IPC_CREAT | 0777;
  printf("%d",sizeof(Heli1_data));
  shmh1_id = shmget(key,sizeof(Heli1_data),flag);
  if(shmh1_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got shared memory of vehicle 1 Position\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli1.Position\n");
  h1 = (Heli1_data *)shmat(shmh1_id,0,0);

  if((int)h1 == -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli1 Position attached.\n");
  *shm_id_ret=shmh1_id;

  printf("Shared memory of h1 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h1->x, h1->y, h1->z);
  return(h1);

}
```

```
Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli2_data *h2;

  printf("Attempt to get Heli2 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh2_id = shmget(key,sizeof(Heli2_data),flag);
  if(shmh2_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli2 Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of Heli2.\n");
  h2 = (Heli2_data *)shmat(shmh2_id,0,0);

  if((int)h2== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli2 attached.\n");
  *shm_id_ret=shmh2_id;

  printf("Shared memory of h2 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h2->x, h2->y, h2->z);
 return(h2);

}

Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Heli3_data *h3;

  printf("Attempt to get Heli3 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh3_id = shmget(key,sizeof(Heli3_data),flag);
  if(shmh3_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli3 Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli3.\n");
```

```
  h3 = (Heli3_data *)shmat(shmh3_id,0,0);

  if((int)h3== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of h3 attached.\n");
  *shm_id_ret=shmh3_id;

  printf("Shared memory of h3 initialized.\n");
 // printf("x = %f, y = %f, z = %f.\n",h3->x, h3->y, h3->z);
 return(h3);

}
Target_data *initialize_target_shared_memory(int key,int *shm_id_ret)
{
  int flag;
  Target_data *t1;

  printf("Attempt to get Target shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmt_id = shmget(key,sizeof(Target_data),flag);
  if(shmt_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Taget Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of vehicle2.\n");
  t1 = (Target_data *)shmat(shmt_id,0,0);

  if((int)t1== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of target attached.\n");
  *shm_id_ret=shmt_id;

    printf("Shared memory of target initialized.\n");
   // printf("x = %f, y = %f, z = %f.\n",t1->x, t1->y, t1->z);
 return(t1);

}

int initialize_target_sem(const char* key)
{
  int flag;
```

```
  int semt_id;
  int sem_id_ret = 1;
  struct sembuf;

   key_t semkey = ftok(key,sem_id_ret);

  if(semkey == (key_t)-1)
    {
      perror("error: ftok() failed\n");
      exit(-1);
    }

  flag = 0666;
  semt_id = semget(semkey,1,flag);
  if(semt_id<0)
    {
      perror("error: semget failed\n");
      exit(-1);
    }

 return(semt_id);
}

void lock_target_sem(int semt_id)
{

  int rc;
  struct sembuf operations[2];
  struct timespec timeout;

 operations[0].sem_num  = 0;
 operations[0].sem_op   = 0;
 operations[0].sem_flg = 0;

 operations[1].sem_num  = 0;
 operations[1].sem_op   = 1;
 operations[1].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,2,&timeout);


}

void release_sem(int semt_id)
{
  int rc;
  struct sembuf operations[2];
  struct timespec timeout;
```

```
 operations[0].sem_num  = 0;
 operations[0].sem_op   = -1;
 operations[0].sem_flg = 0;

 timeout.tv_sec  =  0;
 timeout.tv_nsec = 10000000;

 rc = semtimedop( semt_id, operations,1,&timeout);


}
void cleanup()
{
shmdt(hp1);
shmdt(hp2);
shmdt(hp3);
shmdt(tp);

shmctl(shmh1_id,IPC_RMID,0);
        shmctl(shmh2_id,IPC_RMID,0);
shmctl(shmh3_id,IPC_RMID,0);
shmctl(shmt_id,IPC_RMID,0);

semctl(semt_id, 1, IPC_RMID);
}
```

8.1.6 Formation Control Functions Definition (heliformation.h)

```
/*****************************************************************
* Heliformation.h
* These are routines for solving the dynamics of a system of
* autonomous uav in formation.
* Author: Raul Ordonez & Boakye Dankwa
* Date: May 21st 2007
*
* Some of the library functions of the opensource Autopilot Project was
* used in this code.
*****************************************************************/
#ifndef _HELIFORMATION_H_
#define _HELIFORMATION_H_

#include "Heli.h"

/*************************************************
* UAV state
*************************************************/
typedef struct
{
```

```
Vector<3>Pos;


Vector<3>Vel;


} Xt_def;

/*****************************************************
* UAV state dot
*****************************************************/
typedef struct
{

Vector<3>Pos_dot;


Vector<3>Vel_dot;



} Xt_dot_def;
/*****************************************************
* This computes the Euclaudian norm
*****************************************************/
double norm( Vector<3> & X1,
Vector<3> & Y1
) ;

/*****************************************************
* This computes the derivatives of the
* UAV's, target and filter.
*****************************************************/
void systemDynamics( X_dot_def*  Xdot,  // UAV state dot
Xt_dot_def*  Xtdot, // target state dot
Vector<3>&  Zdot,  // filter state dot
X_def*  X,
X_def*  Xb,
Xt_def* Xt,
Vector<3>& Z,
Vector<3>aij,
Vector<3>bij,
Vector<3>cij,
Vector<6>& data,
double delta_it,
int traj,
double radius,
double speed,
double  heading,
double orientation,
```

```
double slope,
double X0,
double Y0,
double  t);

/*************************************************
* This solves the system states
* using Rk4
*************************************************/
void step( X_dot_def* pXdot,
Xt_dot_def* pXtdot,
Vector<3>& pZdot,
X_def*  pX,// UAV state
Xt_def* pXt,// target state
Vector<3>& pZ,  // filter state
X_def*  pXb,
Vector<3>aij,
Vector<3>bij,
Vector<3>cij,
Vector<6>& data,
double delta_it,
int traj,
double radius,
double speed,
double heading,
double orientation,
double slope,
double X0,
double Y0,
double t,
const double dt);


#endif
```

8.1.7 Formation Control Functions Implementation (heliformation.cpp)

```
/*****************************************************************
* Heliformation.cpp
* These are routines for solving the dynamics of a system of
* autonomous uav's in formation.
* Author: Raul Ordonez & Boakye Dankwa
* Date: May 21st 2007
*****************************************************************/

#include<cmath>

#include "src/timer.h"
#include "Heli.h"
```

```
#include "Heliformation.h"
/*************************************************
* This computes signum function.
*************************************************/
Vector<3> sign(Vector<3> & Y)
{
 Vector<3> Y_out;

for (int i = 0; i <= 2; i++){
if(Y[i] >= 0.0)  Y_out[i] = 1.0;
else Y_out[i] = -1.0;
}

return Y_out;
}

double norm( Vector<3> & X1,
Vector<3> & Y1
)
{
double output;

output = (X1[0] - Y1[0])*(X1[0] - Y1[0])+
(X1[1] - Y1[1])*(X1[1] - Y1[1])+
(X1[2] - Y1[2])*(X1[2] - Y1[2]);

return sqrt(output);
}

void systemDynamics( //outputs
X_dot_def*  Xdot,
Xt_dot_def*  Xtdot,
Vector<3>&  Zdot,
//inputs
X_def*  X,
X_def*  Xb,
Xt_def* Xt,
Vector<3>& Z,
//formation parameters
Vector<3>aij,
Vector<3>bij,
Vector<3>cij,
Vector<6>& data,
double delta_it,
int traj,
double radius,
double speed,
double  heading,
double orientation,
double slope,
double X_init,
```

```
double Y_init,
double t)
{
Vector<3>X_pos(0.0,0.0,0.0);
Vector<3>Xt_pos(0.0,0.0,0.0);
Vector<3>Xb_pos(0.0,0.0,0.0);
Vector<3>target_pos_dot(0.0,0.0,0.0);
Vector<3>grad_J(0.0,0.0,0.0);

Vector<3>surface(0.0,0.0,0.0);
Vector<3>U_force(0.0,0.0,0.0);
Vector<3>U;


Vector<3>Xv1(0.0,0.0,0.0);
Vector<3>Xv2(0.0,0.0,0.0);
Vector<3>temp1(0.0,0.0,0.0);
Vector<3>temp2(0.0,0.0,0.0);

/*****************************************************
* Please refere to thesis for definitions
* of these parameters.
*****************************************************/

Matrix<3,3>b_bar;
//distance between ith agent and kth reference point.
double h_ik = delta_it*sqrt(2.0);

double Kv = 10.0;

double  m_u_bar = 10.0;//1.5;
double m_o_bar = 5.0;//0.5;
double alpha = 0.01;
double beta = 5.0;//1.2;
double mu = 0.1;
double  epsilon = 100.0;//1.0;
double bb = 10.0;
double f_bar = 1.0;
double d = 12.0;
double  J_bar = alpha*alpha * sqrt(2*d*d) + alpha*(5.5*beta);
double J_bar2  = 2.0*beta/mu;
double Kt = 10.0;
double Kf = 500.0;

double x_o = X_init;
double y_o = Y_init;

double freq = speed/(2.0*pi*radius);
double xt_d = 0.0;
double yt_d = 0.0;
double  u0;
```

```
double J;
/*****************************************************
* Computing the straight line trajectory
* for the target.
*****************************************************/
if(traj == 1){
if( (heading > 0.0 )&&(heading <= pi/2.0) ){
xt_d = 1.0;
yt_d = slope;}
if( (heading >= pi/2.0)&&(heading < pi ) ){
xt_d = 1.0;
yt_d = slope;}
if( (heading >= -pi/2.0 )&&(heading < 0.0 ) ){
xt_d = -1.0;
yt_d = -slope;}
if( (heading > -pi )&&(heading <= -pi/2.0) ){
xt_d = -1.0;
yt_d = -slope;}

if (heading == 0.0){
xt_d = 0.0;
yt_d = 1.0;}
if ((heading == pi)||(heading ==-pi)){
xt_d = 0.0;
yt_d = -1.0;}

}
/*****************************************************
* Computing the circular trajectory
* for the target.
*****************************************************/

if(traj == 3){
xt_d = -2.0*pi*radius*freq*sin(2.0*pi*freq*t);
yt_d = 2.0*pi*radius*freq*cos(2.0*pi*freq*t);
Kv = 0.0;}

/*****************************************************
* Computing the "reference frame".
*****************************************************/
if(orientation == 0.0)
{
temp1[0] = 0.0;
temp1[1] = 1.0;
temp2[0] = 0.0;
temp2[1] = -1.0;
}
if(orientation == 90.0)
{
temp1[0] = -1.0;
temp1[1] = 0.0;
```

```
temp2[0] = 1.0;
temp2[1] = 0.0;
}
if(orientation == -90.0)
{
temp1[0] = 1.0;
temp1[1] = 0.0;
temp2[0] = -1.0;
temp2[1] = 0.0;
}
if( orientation == 180.0 || orientation == -180.0 )
{
temp1[0] = 0.0;
temp1[1] = -1.0;
temp2[0] = 0.0;
temp2[1] = 1.0;
}
if(orientation > 0.0 && orientation < 90.0)
{
temp1[0] = -sin(orientation*pi/180.0);
temp1[1] =  cos(orientation*pi/180.0);
temp2[0] =  sin(orientation*pi/180.0);
temp2[1] = -cos(orientation*pi/180.0);
}
if(orientation > 90.0 && orientation < 180.0)
{
temp1[0] = -sin(orientation*pi/180.0);
temp1[1] =  cos(orientation*pi/180.0);
temp2[0] = -cos(orientation*pi/180.0);
temp2[1] =  sin(orientation*pi/180.0);
}
if(orientation > -90.0 && orientation < 0.0)
{
temp1[0] =  cos(orientation*pi/180.0);
temp1[1] =  -sin(orientation*pi/180.0);
temp2[0] =  -cos(orientation*pi/180.0);
temp2[1] =  sin(orientation*pi/180.0);
}
if(orientation > -180.0 && orientation < -90.0)
{
temp1[0] = -cos(orientation*pi/180.0);
temp1[1] =  sin(orientation*pi/180.0);
temp2[0] =  sin(orientation*pi/180.0);
temp2[1] = -cos(orientation*pi/180.0);
}
// reference frame
b_bar.col(1,temp1);
b_bar.col(2,temp2);

X_pos  =  X->Pos;
Xt_pos = Xt->Pos;
```

```
Xb_pos = Xb->Pos;

// positions of reference points
Xv1 = Xt_pos + b_bar.col(1)*delta_it;
Xv2 = Xt_pos + b_bar.col(2)*delta_it;

// Make sure we're computing in 2D.
X_pos[2] = 0.0;
Xb_pos[2] = 0.0;
Xt_pos[2] = 0.0;
Xv1[2] = 0.0;
Xv2[2] = 0.0;
Zdot[2] = 0.0;
Z[2] = 0.0;
X->Pos[2] = 0.0;
X->Vel[2] = 0.0;

/*****************************************************************
* Compute the gradient.
*****************************************************************/
grad_J = (X_pos - Xt_pos)*2.0*Kt*(norm(X_pos,Xt_pos) *
 norm(X_pos,Xt_pos) -
delta_it*delta_it) +
((X_pos - Xb_pos)*aij[1] - (X_pos - Xb_pos)*bij[1]*
       exp(-norm(X_pos,Xb_pos) *
       norm(X_pos,Xb_pos)
/cij[1]))*Kf +
((X_pos - Xv1)*(norm(X_pos,Xv1) *
norm(X_pos,Xv1) -
h_ik*h_ik)  + (X_pos - Xv2)*(norm(X_pos,Xv2) *
       norm(X_pos,Xv2) -
h_ik*h_ik))*2.0*Kv;
/*****************************************************************
* Compute the potential.
*****************************************************************/
J = 0.0*(0.5*pow((norm(X_pos,Xt_pos)*norm(X_pos,Xt_pos)-
       delta_it*delta_it),2.0) +
    0.5*pow((norm(Xb_pos,Xt_pos)*norm(Xb_pos,Xt_pos)-
     delta_it*delta_it),2.0))+
Kf*(aij[1]/2.0*pow(norm(X_pos,Xb_pos),2.0) +
bij[1]*cij[1]/2.0*exp(-norm(X_pos,Xb_pos) *
       norm(X_pos,Xb_pos)
/cij[1])) +
0.0*(0.5*pow((norm(X_pos,Xv1)*norm(X_pos,Xv1)- h_ik*h_ik),2.0) +
            0.5*pow((norm(X_pos,Xv2)*norm(X_pos,Xv2)- h_ik*h_ik),2.0) +
    0.5*pow((norm(Xb_pos,Xv1)*norm(Xb_pos,Xv1)- h_ik*h_ik),2.0) +
            0.5*pow((norm(Xb_pos,Xv2)*norm(Xb_pos,Xv2)- h_ik*h_ik),2.0) );

/*************************************************
* Compute the filter derivatives.
*************************************************/
```

```
Zdot = (Z*(-1.0) + sign(grad_J)*beta)*1.0/mu;


target_pos_dot[0] = speed*xt_d;//0.5;
target_pos_dot[1] = speed*yt_d;//20.0*sin(2.0*t);
target_pos_dot[2] = 0.0;

if(traj == 3){
target_pos_dot[0] = xt_d;//0.5;
target_pos_dot[1] = yt_d;//20.0*sin(2.0*t);
target_pos_dot[2] = 0.0; }


Xtdot->Pos_dot = target_pos_dot;

/*****************************************************
* Compute sliding surface.
*****************************************************/
surface = X->Vel + grad_J * alpha +  Z;

u0 = m_u_bar * (1.0/m_o_bar * f_bar + alpha*J_bar + J_bar2 + epsilon);
//printf("\n u0: %f",u0);

U_force[0] = -u0 * tanh(bb * surface[0]);
  U_force[1] = -u0 * tanh(bb * surface[1]);
U_force[2] = -u0 * tanh(bb * surface[2]);

U = U_force;
/*****************************************************
* Compute the derivatives of point mass model.
*****************************************************/
       Heli( Xdot,
X,
U,
t);

Xdot->Pos_dot[2] = 0.0;
         Xdot->Vel_dot[2] = 0.0;
        //reusing empty arrays
   data[0] = U[0];
data[1] = U[1];
data[2] = grad_J[0];
data[3] = grad_J[1];
data[4] = J;
data[5] = u0;


}
/*****************************************************
* Solve system states using the RK4 algorithm.
*****************************************************/
```

```
void step( X_dot_def* pXdot,
Xt_dot_def* pXtdot,
Vector<3>& pZdot,
X_def*  pX,
Xt_def* pXt,
Vector<3>& pZ,
X_def*  pXb,
Vector<3>aij,
Vector<3>bij,
Vector<3>cij,
Vector<6>& data,
double delta_it,
int traj,
double radius,
double speed,
double heading,
double  orientation,
double slope,
double X_init,
double Y_init,
double t,
double dt)

{
// to run the derivative function
X_def X;
Xt_def Xt;
Vector<3>Z;

// to backup the input state
X_def X0;
Xt_def Xt0;
Vector<3>Z0;

// to run the derivative function
X_dot_def Xdot;
Xt_dot_def Xtdot;
Vector<3>Zdot;

int i;
double k_X_1[6];
double k_X_2[6];
double k_X_3[6];
double k_X_4[6];

double k_Xt_1[6];
double k_Xt_2[6];
double k_Xt_3[6];
double k_Xt_4[6];

double k_Z_1[3];
```

```
double k_Z_2[3];
double k_Z_3[3];
double k_Z_4[3];
// backing up the current input state
X0.Pos = pX->Pos;
X0.Vel = pX->Vel;

Xt0.Pos = pXt->Pos;
Xt0.Vel = pXt->Vel;

Z0 = pZ;

// Well, lets get started with the first step
// first, make the X state value
X.Pos = X0.Pos;
X.Vel = X0.Vel;

Xt.Pos = Xt0.Pos;
Xt.Vel = Xt0.Vel;

Z = Z0;


// run the function and get the Xdot values
systemDynamics( //outputs
&Xdot,
&Xtdot,
Zdot,
//inputs
&X,
pXb,
&Xt,
Z,
//formation parameters
aij,
bij,
cij,
data,
delta_it,
traj,
radius,
speed,
heading,
orientation,
slope,
X_init,
Y_init,
t);
for(i=0; i<3; i++)
{
// save the k step
```

```
k_X_1[i]  = dt*Xdot.Pos_dot[i];
k_X_1[i+3]  = dt*Xdot.Vel_dot[i];

k_Xt_1[i]  = dt*Xtdot.Pos_dot[i];
k_Xt_1[i+3]  = dt*Xtdot.Vel_dot[i];

k_Z_1[i]  = dt*Zdot[i];

// make the state value for the next step
X.Pos[i] = X0.Pos[i] + k_X_1[i]/2.0;
X.Vel[i]  = X0.Vel[i] + k_X_1[i+3]/2.0;

Xt.Pos[i] = Xt0.Pos[i] + k_Xt_1[i]/2.0;
Xt.Vel[i]  = Xt0.Vel[i] + k_Xt_1[i+3]/2.0;

Z[i] = Z0[i] + k_Z_1[i]/2.0;

}

// save the Xdot values for output
pXdot->Pos_dot = Xdot.Pos_dot;
pXdot->Vel_dot = Xdot.Vel_dot;

pXtdot->Pos_dot = Xtdot.Pos_dot;
pXtdot->Vel_dot = Xtdot.Vel_dot;

pZdot = Zdot;

// run the function and get the Xdot values
systemDynamics( //outputs
&Xdot,
&Xtdot,
Zdot,
//inputs
&X,
pXb,
&Xt,
Z,
//formation parameters
aij,
bij,
cij,
data,
delta_it,
traj,
radius,
speed,
heading,
orientation,
slope,
X_init,
```

```
Y_init,
t);
for(i=0; i<3; i++)
{
// save the k step
k_X_2[i]  = dt*Xdot.Pos_dot[i];
k_X_2[i+3]  = dt*Xdot.Vel_dot[i];

k_Xt_2[i]  = dt*Xtdot.Pos_dot[i];
k_Xt_2[i+3]  = dt*Xtdot.Vel_dot[i];

k_Z_2[i]  = dt*Zdot[i];

// make the state value for the next step
X.Pos[i] = X0.Pos[i] + k_X_2[i]/2.0;
X.Vel[i]  = X0.Vel[i] + k_X_2[i+3]/2.0;

Xt.Pos[i] = Xt0.Pos[i] + k_Xt_2[i]/2.0;
Xt.Vel[i]  = Xt0.Vel[i] + k_Xt_2[i+3]/2.0;

Z[i] = Z0[i] + k_Z_2[i]/2.0;

}

// run the function and get the Xdot values
systemDynamics( //outputs
&Xdot,
&Xtdot,
Zdot,
//inputs
&X,
pXb,
&Xt,
Z,
//formation parameters
aij,
bij,
cij,
data,
delta_it,
traj,
radius,
speed,
heading,
orientation,
slope,
X_init,
Y_init,
t);
for(i=0; i<3; i++)
{
```

```
// save the k step
k_X_3[i]   = dt*Xdot.Pos_dot[i];
k_X_3[i+3]  = dt*Xdot.Vel_dot[i];

k_Xt_3[i]  = dt*Xtdot.Pos_dot[i];
k_Xt_3[i+3]  = dt*Xtdot.Vel_dot[i];

k_Z_3[i]  = dt*Zdot[i];

// make the state value for the next step
X.Pos[i] = X0.Pos[i] + k_X_3[i]/2.0;
X.Vel[i]  = X0.Vel[i] + k_X_3[i+3]/2.0;

Xt.Pos[i] = Xt0.Pos[i] + k_Xt_3[i]/2.0;
Xt.Vel[i]  = Xt0.Vel[i] + k_Xt_3[i+3]/2.0;

Z[i] = Z0[i] + k_Z_3[i]/2.0;

}

// run the function and get the Xdot values
systemDynamics( //outputs
&Xdot,
&Xtdot,
Zdot,
//inputs
&X,
pXb,
&Xt,
Z,
//formation parameters
aij,
bij,
cij,
data,
delta_it,
traj,
radius,
speed,
heading,
orientation,
slope,
X_init,
Y_init,
t);
for(i=0; i<3; i++)
{
// save the k step
k_X_4[i]  = dt*Xdot.Pos_dot[i];
k_X_4[i+3]  = dt*Xdot.Vel_dot[i];
```

```
k_Xt_4[i]   = dt*Xtdot.Pos_dot[i];
k_Xt_4[i+3]   = dt*Xtdot.Vel_dot[i];

k_Z_4[i]   = dt*Zdot[i];
}



// assemble the final result for the state propogation
for(i=0; i<3; i++)
{
pX->Pos[i]   = X0.Pos[i] + k_X_1[i]/6.0 + k_X_2[i]/3.0 +
                           k_X_3[i]/3.0 + k_X_4[i]/6.0;
pX->Vel[i]   = X0.Vel[i] + k_X_1[i+3]/6.0 + k_X_2[i+3]/3.0 +
                           k_X_3[i+3]/3.0 + k_X_4[i+3]/6.0;

pXt->Pos[i]   = Xt0.Pos[i] + k_Xt_1[i]/6.0 + k_Xt_2[i]/3.0 +
                             k_Xt_3[i]/3.0 + k_Xt_4[i]/6.0;
pXt->Vel[i]   = Xt0.Vel[i] + k_Xt_1[i+3]/6.0 + k_Xt_2[i+3]/3.0 +
                             k_Xt_3[i+3]/3.0 + k_Xt_4[i+3]/6.0;

pZ[i]   = Z0[i] + k_Z_1[i]/6.0 + k_Z_2[i]/3.0 + k_Z_3[i]/3.0 +
                  k_Z_4[i]/6.0;

}


 }
```

8.2 Code For The Graphical User Interface

```
**********************main.cpp*********************************
#include <qapplication.h>

#include "maingui.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainGui *gui = new MainGui;
    app.setMainWidget(gui);
    gui->show();
    return app.exec();
}
/*************************************************************/
/*************************************maingui.h**************
* This code creates the main window.
* Author: Boakye Dankwa
* Data: June 2007
*************************************************************/
#ifndef MAINGUI_H
```

```
#define MAINGUI_H

#include <qwidget.h>
#include "Missions/missionswindowImpl.h"

class QLabel;
class QPushButton;
class Plotter;

class MainGui : public QWidget
{
    Q_OBJECT
public:
     MainGui(QWidget *parent = 0, const char *name = 0);
signals:
     void openMissionsWindow();
     void runAlgorithm();
     void abortMission();
     void exiting();

private slots:
      void missionClicked();
     void runClicked();
     void abortClicked();
     void closeClicked();
     void enableRunButton();
     void enableAbortButton();
     void showWarning();
     void showCriticalWarning();

private:

/*********************************
*  labels
*********************************/
  QLabel *missionPlotLabel;
QLabel *formationPlotLabel;
QLabel *warningLabel;
QLabel *statusLabel;
QLabel * lcd1Label;
QLabel * lcd2Label;
QLabel * lcd3Label;
QLabel * lcd4Label;
QLabel * lcd5Label;
/*********************************
*   buttons
*********************************/
QPushButton *missionButton;
QPushButton *runButton;
QPushButton *abortButton;
QPushButton *exitButton;
```

```
/***********************************
*   LCD displays
***********************************
QLCDNumber *heli1AltLCD;
QLCDNumber *heli2AltLCD;
QLCDNumber *heli3AltLCD;
QLCDNumber *mindeltaijLCD;
QLCDNumber *maxdeltaijLCD;
QLCDNumber *timeLCD;

Plotter * missionPlot;
missionswindowImpl *mWindow;
};

#endif
/*****************************************************************/
/******************maingui.h*********************************
* This code creates the main window.
* Author: Boakye Dankwa
* Data: June 2007
*****************************************************************/

#include <qlabel.h>
#include <qlayout.h>
#include <qpushbutton.h>
#include <qfile.h>

#include "plotter.h"
#include "maingui.h"

MainGui::MainGui(QWidget *parent, const char *name)
     : QWidget(parent, name)
{
  setCaption(tr("Formation Control"));
  setGeometry(50,50,800,750);
  setMinimumSize(800,750);
  setMaximumSize(800,750);

 /*********************************************************
 *   Draw the labels
 *********************************************************/
warningLabel = new QLabel(tr("Warnings:"), this);
statusLabel = new QLabel(tr("No UAV connected"), this);
  lcd1Label    = new QLabel(tr("UAV1 Alt"),this);
  lcd2Label    = new QLabel(tr("UAV2 Alt"),this);
lcd3Label    = new QLabel(tr("UAV3 Alt"),this);
  lcd4Label    = new QLabel(tr("min sep"),this);
lcd5Label    = new QLabel(tr("max Sep"),this);

 /*********************************************************
 *   Draw the buttons
```

```
 ********************************************************/
missionButton = new QPushButton(tr("&Mission"), this);
  missionButton->setMaximumSize(4000,4000);
missionButton->setDefault(true);
missionButton->setEnabled(true);

runButton = new QPushButton(tr("&Run"), this);
  runButton->setMaximumSize(4000,4000);
runButton->setEnabled(false);

abortButton = new QPushButton(tr("&Abort"), this);
  abortButton->setMaximumSize(4000,4000);
  abortButton->setPaletteBackgroundColor(red.light());
abortButton->setEnabled(false);

  exitButton = new QPushButton(tr("&Exit"), this);
  exitButton->setMaximumSize(4000,4000);
  missionPlot = new Plotter(this);

 /*********************************************************
 *   Create a 'mission window' object and make it a child
 *  of the 'main' window
 *********************************************************/
  mWindow = new missionswindowImpl(this);


/*********************************************************
 *   Create the LCD objects and set them on
 *   the 'main' window
 *********************************************************/
  heli1AltLCD =  new QLCDNumber( 5,this);
  heli2AltLCD =  new QLCDNumber( 5,this);
  heli3AltLCD =  new QLCDNumber( 5,this);
  mindeltaijLCD =  new QLCDNumber( 5,this);
maxdeltaijLCD =  new QLCDNumber( 5,this);
  timeLCD =  new QLCDNumber( 7,this);

mWindow->setLCD(heli1AltLCD, heli2AltLCD, heli3AltLCD,
                mindeltaijLCD, maxdeltaijLCD, timeLCD);

heli1AltLCD->setFrameStyle(QFrame::WinPanel | QFrame::Plain);
  heli2AltLCD->setFrameStyle(QFrame::WinPanel | QFrame::Plain);
  heli3AltLCD->setFrameStyle(QFrame::WinPanel | QFrame::Plain);
mindeltaijLCD->setFrameStyle(QFrame::WinPanel | QFrame::Plain);
maxdeltaijLCD->setFrameStyle(QFrame::WinPanel | QFrame::Plain);
timeLCD->setFrameStyle(QFrame::MenuBarPanel | QFrame::Plain );
timeLCD->setMidLineWidth(0);

heli1AltLCD->setSegmentStyle(QLCDNumber::Flat);
  heli2AltLCD->setSegmentStyle(QLCDNumber::Flat);
  heli3AltLCD->setSegmentStyle(QLCDNumber::Flat);
```

```
mindeltaijLCD->setSegmentStyle(QLCDNumber::Flat);
maxdeltaijLCD->setSegmentStyle(QLCDNumber::Flat);
timeLCD->setSegmentStyle(QLCDNumber::Flat);

/*******************************************************
 *   Establish the signal-slot connections
  *******************************************************/
connect(missionButton, SIGNAL(clicked()),this, SLOT(missionClicked()));

  connect(runButton, SIGNAL(clicked()),this, SLOT(runClicked()));

connect(abortButton, SIGNAL(clicked()),this, SLOT(abortClicked()));

connect(exitButton, SIGNAL(clicked()), this, SLOT(closeClicked()));

connect(exitButton, SIGNAL(clicked()), this, SLOT(close()));

connect(mWindow,SIGNAL(connected()), this, SLOT(enableRunButton()));

connect(mWindow,SIGNAL(connected()),this, SLOT(enableAbortButton()));

connect(mWindow,SIGNAL(warning()), this, SLOT(showWarning()));

connect(mWindow,SIGNAL(aborting()), this, SLOT(showCriticalWarning()));

/*****************************************
 *   Arrange all the created objects on the
 *  'main' window
 *****************************************/

  QVBoxLayout *topLeftLayout = new QVBoxLayout;
  topLeftLayout->setResizeMode(QLayout::Fixed);
  topLeftLayout->addWidget(missionPlot);


    QVBoxLayout *topRightLayout = new QVBoxLayout;
    topRightLayout->addWidget(missionButton);
    topRightLayout->addWidget(runButton);
    topRightLayout->addWidget(abortButton);
    topRightLayout->addWidget(exitButton);
    topRightLayout->addWidget(lcd1Label);
    topRightLayout->addWidget(heli1AltLCD);
    topRightLayout->addWidget(lcd2Label);
    topRightLayout->addWidget(heli2AltLCD);
    topRightLayout->addWidget(lcd3Label);
    topRightLayout->addWidget(heli3AltLCD);
    topRightLayout->addWidget(lcd4Label);
    topRightLayout->addWidget(mindeltaijLCD);
    topRightLayout->addWidget(lcd5Label);
    topRightLayout->addWidget(maxdeltaijLCD);
```

```
    QHBoxLayout *topLayout = new QHBoxLayout;
    topLayout->addLayout(topLeftLayout);
    topLayout->addLayout(topRightLayout);

    QVBoxLayout *topButtomLayout = new QVBoxLayout;
    topButtomLayout->addWidget(timeLCD);
    topButtomLayout->addWidget(warningLabel);

    QHBoxLayout *buttomButtomLayout = new QHBoxLayout;
    buttomButtomLayout->addWidget(statusLabel);

    QVBoxLayout *buttomLayout = new QVBoxLayout;
    buttomLayout->addLayout(topButtomLayout);
    buttomLayout->addLayout(buttomButtomLayout);

    QVBoxLayout *mainLayout = new QVBoxLayout(this);
    mainLayout->setMargin(11);
    mainLayout->setSpacing(6);
    mainLayout->addLayout(topLayout);
    mainLayout->addLayout(buttomLayout);

}
/***************************************
*  This function is called when the
*  'missions' button is clicked.
***************************************
void MainGui::missionClicked()
{
   mWindow->show();
   mWindow->setGraph(missionPlot);
}
/***************************************
*  This function is called when the
*  'run' button is clicked.
***************************************

void MainGui::runClicked(){

  runButton->setEnabled(false);
  missionButton->setEnabled(false);
  exitButton->setEnabled(false);
  mWindow->runAlgorithm();
}
/***************************************
*  This function is called when the
*  'abort' button is clicked.
***************************************
void MainGui::abortClicked()
{
  QString st = "disconnected.....choppers enroute to failsafe position";
```

```
  mWindow->abortMission();
  missionButton->setEnabled(true);
  exitButton->setEnabled(true);
  runButton->setEnabled(false);
  abortButton->setEnabled(false);


   statusLabel->setText(st);
}
/****************************************
*  This function is called when the
*  main window is closed or 'exit'
*  is clicked.
****************************************
void MainGui::closeClicked()
{
   mWindow->haltAll();
}
/****************************************
*  This function is called when data
*  from the 'missions window' is
*  accepted.
****************************************
void MainGui::enableRunButton(){

   runButton->setEnabled(true);
   QString st = "Ready";
   statusLabel->setText(st);

}
/****************************************
*  This function is called when the
*  'abort' button is clicked.
****************************************
void MainGui::enableAbortButton()
{

   abortButton->setEnabled(true);
   missionButton->setEnabled(false);
   exitButton->setEnabled(false);
}
/****************************************
*  This function is called when the
*  the UAVs are getting too close to each
*  other
****************************************
void MainGui::showWarning()
{
QString wr = warningLabel->text();
wr.append(" current separation less than required!");
 warningLabel->setPaletteForegroundColor(red);
```

```
 warningLabel->setText(wr);
}
/*************************************
*  This function is called just
*  before 'safe mode'.
*************************************
void MainGui::showCriticalWarning()
{
QString wr = warningLabel->text();
QString st = "disconnected";
wr.append(" current separation less than minimum allowable....mission aborted!");
 warningLabel->setPaletteForegroundColor(red);
 warningLabel->setText(wr);
        statusLabel->setText(st);
 abortButton->setEnabled(false);
        exitButton->setEnabled(true);

}
/*****************************************************************************/
/***************************missionswindowImpl.h*******************
* This file implements the functionality of the Missions Window
* Author: Boakye Dankwa
*
* Date: June 2007
*****************************************************************************/
#ifndef MISSIONSWINDOWIMPL_H
#define MISSIONSWINDOWIMPL_H
#include "missionswindow.h"
#include "Plotters/Real_time/plotter.h"
#include <qlayout.h>
#include <qgroupbox.h>
#include <qlineedit.h>
#include <qbuttongroup.h>
#include <qspinbox.h>
#include <qpushbutton.h>
#include <qradiobutton.h>
#include <qlabel.h>
#include <qlcdnumber.h>
#include <qfile.h>
#include <qtimer.h>
#include <qprocess.h>
#include <qslider.h>
extern "C" {
#include "control/cccs.h"
}

//Structure for storing formation parameters
typedef struct
{

int trajectory;
```

```
double altitude;
double formation;
double radius;
int speed;
double heading;
double orientation;
double slope;
double X0;
double Y0;

} INPUTS;

//Position structure
typedef struct
{

double north;
double east;
double down;

} POSITION;

class missionswindowImpl : public missionsWindow
{
    Q_OBJECT

public:
    missionswindowImpl( QWidget* parent = 0, const char* name = 0,
                bool modal = FALSE, WFlags fl = 0 ) ;
    ~missionswindowImpl();

signals: void connected();
   void warning();
   void aborting();

public slots:
    void enableUavLineEdit();
    void enableConnectButton();
    void connectButtonClicked();
    void runAlgorithm();
    void setGraph(Plotter*);
    void setLCD(QLCDNumber*,QLCDNumber*,QLCDNumber*,
                QLCDNumber*, QLCDNumber*, QLCDNumber* );
    void displayHeli1Data();
    void displayHeli2Data();
    void displayTargetData();
    void drawStraightLinePath();
    void drawWaypointCurve();
    void drawCirclePath();
    void drawPath();
    void setSafeBounds(const double);
```

```
    void plotPosition(double,double,int);
    void initializeSharedMemory();
    void do_warning();
    void abortMission();
    void haltAll();

    Heli1_data *initialize_heli1pos_shared_memory(int key,int *shm_id_ret);
    Heli2_data *initialize_heli2pos_shared_memory(int key,int *shm_id_ret);
    Heli3_data *initialize_heli3pos_shared_memory(int key,int *shm_id_ret);
    Target_data *initialize_target_shared_memory(int key,int *shm_id_ret);

private:

const char * uav1IP ;
      const char * uav2IP;
      const char * uav3IP;

      int uav1Port;
      int uav2Port;
      int uav3Port;

     double delta_ij[4][4];
double delta_it[4];


POSITION  heli1_x0;
      POSITION  heli2_x0;
      POSITION  heli3_x0;

QLCDNumber* LCD1;
QLCDNumber* LCD2;
QLCDNumber* LCD3;
QLCDNumber* LCD4;
QLCDNumber* LCD5;
QLCDNumber* LCD6;


      QTimer * updateTimer ;

      Plotter* graph;

double separation;
double minseparation;
     static const double criticalSeparation;
      int missionStatus;
      int warningCounter;
int numberOfUAV;

INPUTS inputs;
```

```
QProcess* procT;
QProcess* proc1;
QProcess* proc2;
QProcess* proc3;

};

#endif // MISSIONSWINDOWIMPL_H
*********************************************************************
/***********missionswindowImpl.cpp*********************************/
#include "missionswindowImpl.h"
#include <qvalidator.h>
#include <playerclient.h>
#include <qprogressdialog.h>
#include <qmessagebox.h>
#include <qtextstream.h>
#include <timer.h>
#define  BUFSIZE 256
#define  UPDATETIM 0.5
#define pi 3.1415926535

Heli1_data *hp1;
Heli2_data *hp2;
Heli3_data *hp3;
Target_data *tp;

int shmh1_id;
int shmh2_id;
int shmh3_id;
int shmt_id;

const double missionswindowImpl::criticalSeparation = 6.0;

missionswindowImpl::missionswindowImpl( QWidget* parent,
const char* name, bool modal, WFlags fl )
    : missionsWindow( parent, name, modal, fl )
{

updateTimer = new QTimer(this);
QRegExp  IPregExp("[0-9]{1,3}[.][0-9]{1,3}[.][0-9]{1,3}[.][0-9]{1,3}");
                  uav1LineEdit->setValidator(new QRegExpValidator(IPregExp,this));
                  uav2LineEdit->setValidator(new QRegExpValidator(IPregExp,this));
                  uav3LineEdit->setValidator(new QRegExpValidator(IPregExp,this));

    QRegExp  PortExp("[7][0][0][0-2]");
uav1PortLineEdit->setValidator(new QRegExpValidator(PortExp,this));
uav2PortLineEdit->setValidator(new QRegExpValidator(PortExp,this));
uav3PortLineEdit->setValidator(new QRegExpValidator(PortExp,this));

    QRegExp  WaypointExp("*.txt");
waypointFileLineEdit->setValidator(new QRegExpValidator(WaypointExp,this));
```

```
    QDoubleValidator  *altExp = new   QDoubleValidator( 0.0,10.0,1,this);
                       altLineEdit->setValidator(altExp);

    QDoubleValidator  *orientationExp = new  QDoubleValidator( -180.0,180.0,0,this);
       orientationLineEdit->setValidator(orientationExp);

     QDoubleValidator  *positionExp = new   QDoubleValidator( -5000.0,5000.0,4,this);
                       slNLineEdit->setValidator(positionExp);
                       slELineEdit->setValidator(positionExp);
                       circleNLineEdit->setValidator(positionExp);
                       circleELineEdit->setValidator(positionExp);

    QDoubleValidator  *distanceExp = new QDoubleValidator(0.0,5000.0,4,this);
        slLLineEdit->setValidator(distanceExp);
        circleRLineEdit->setValidator(distanceExp);

      QDoubleValidator  *headingExp = new   QDoubleValidator( -180.0,180.0,4,this);
                       slHLineEdit->setValidator(headingExp);



     connectButton->setDisabled(true);

// signals and slots connections
    connect( uavSpinBox, SIGNAL( valueChanged(int) ), this,
    SLOT( enableUavLineEdit() ));
    connect( connectButton, SIGNAL( clicked() ),  this,
    SLOT( connectButtonClicked() ));
    connect( uav1LineEdit, SIGNAL(textChanged(const QString&)), this,
     SLOT( enableConnectButton()));
    connect( uav2LineEdit, SIGNAL(textChanged(const QString&)), this,
     SLOT( enableConnectButton()));
    connect( connectButton,SIGNAL(clicked()),   this, SLOT(drawPath()));
    connect( updateTimer,SIGNAL(timeout()), this, SLOT(displayHeli1Data()));
    connect( updateTimer,SIGNAL(timeout()), this, SLOT(displayHeli2Data()));
    connect( updateTimer,SIGNAL(timeout()), this, SLOT(displayTargetData()));
    connect( this,SIGNAL(aborting()),this,SLOT(abortMission()));


    inputs.trajectory = 1;
inputs.altitude = -20.0;
inputs.formation = 10.0;
    inputs.radius = 0.0;
inputs.speed = 0.0;
inputs.heading = 0.0;
inputs.orientation = 0.0;
    inputs.slope = 0.0;
    inputs.X0 = 0.0;
    inputs.Y0 = 0.0;
```

```
heli1_x0.north = 0.0;
heli1_x0.east = 0.0;
heli1_x0.down = 0.0;
    heli2_x0.north = 0.0;
heli2_x0.east = 0.0;
heli2_x0.down = 0.0;
    heli3_x0.north = 0.0;
heli3_x0.east = 0.0;
heli3_x0.down = 0.0;

numberOfUAV = 0;
missionStatus = 1;
warningCounter = 0;

}

/*
 *  Destroys the object and frees any allocated resources
 */
missionswindowImpl::~missionswindowImpl()
{
    // no need to delete child widgets, Qt does it all for us
}
void missionswindowImpl::setGraph(Plotter* missionPlot)
{
  graph = missionPlot;
}
void missionswindowImpl::setLCD(QLCDNumber * lcd1,
QLCDNumber * lcd2,
QLCDNumber * lcd3,
QLCDNumber * lcd4,
QLCDNumber * lcd5,
QLCDNumber * lcd6)
{
   LCD1 = lcd1;
   LCD2 = lcd2;
   LCD3 = lcd3;
   LCD4 = lcd4;
   LCD5 = lcd5;
   LCD6 = lcd6;

   LCD3->display("--");
   LCD5->display("--");
   LCD6->display("0:00:00");
}

void missionswindowImpl::displayHeli1Data()
{
```

```
separation = (hp1->position.x - hp2->position.x)*
(hp1->position.x - hp2->position.x) +
                        (hp1->position.y - hp2->position.y)*
(hp1->position.y - hp2->position.y) +
(hp1->position.z - hp2->position.z)*
(hp1->position.z - hp2->position.z);

        separation = sqrt(separation);

if ((missionStatus == 1 )&& (separation <= criticalSeparation))
emit aborting();




LCD1->display(hp1->position.z);
  LCD4->display(separation);


  plotPosition(hp1->position.x,hp1->position.y,0);


static int k = 0;
static int t = 0;
  static double  trail_coord[4000];
  CurveData trail_data;

int sec;
int min;
  int hr;

sec = t;
hr = sec/(60*60);
sec = sec - hr*60*60;
        min = sec/60;
sec = sec - min*60;

QString timestr = QString("%1:%2:%3").arg(hr).arg(min).arg(sec);

LCD6->display(timestr);
        trail_coord[k] = hp1->position.x;
        trail_coord[k+1] = hp1->position.y;

    for (int n = 0; n<k;++n) {
   trail_data.push_back(trail_coord[n]);
}

if( k < 40000 )graph->setTrailData(0, trail_data);
```

```
k = k +2;
        t++;
}
void missionswindowImpl::displayHeli2Data()
{
LCD2->display(hp2->position.z);

        plotPosition(hp2->position.x,hp2->position.y,1);

static int k = 0;
        static double  trail_coord[4000];
        CurveData trail_data;

        trail_coord[k] = hp2->position.x;
        trail_coord[k+1] = hp2->position.y;

     for (int n = 0; n<k;++n) {
    trail_data.push_back(trail_coord[n]);
}

if( k < 4000 )graph->setTrailData(1, trail_data);


k = k +2;

}

void missionswindowImpl::displayTargetData()
{
         plotPosition(tp->position.x,tp->position.y,5);
}

void missionswindowImpl::setSafeBounds(const double safe1)
{
  minseparation = safe1;
}

void missionswindowImpl::do_warning()
{
if(warningCounter < 1){
switch( QMessageBox::warning(0, "fcontrol",
"one or more UAVs is not connected.\n"
                        "What do you want to do?.\n",
QMessageBox::Abort | QMessageBox::Default,
QMessageBox::Ignore | QMessageBox::Escape) ) {
case  QMessageBox::Abort:
if(missionStatus == 1)abortMission();
break;
case  QMessageBox::Ignore:
break;
```

```
} warningCounter++;return;
}

}

void missionswindowImpl::enableUavLineEdit()
{

    if ((uavSpinBox->value()) != 2 ){
uav3LineEdit->setEnabled(TRUE);
uav3PortLineEdit->setEnabled(TRUE);
                uav13LineEdit->setEnabled(TRUE);
uav23LineEdit->setEnabled(TRUE);}
    else
    { uav3LineEdit->setEnabled(FALSE);
uav3PortLineEdit->setEnabled(FALSE);
                uav13LineEdit->setEnabled(FALSE);
uav23LineEdit->setEnabled(FALSE);}
}

/*
 * public slot
 */
void missionswindowImpl::enableConnectButton()
{
   if((uavSpinBox->value())!=2){
    connectButton->setEnabled( (uav1LineEdit->hasAcceptableInput()) &&
         (uav1PortLineEdit->hasAcceptableInput())&&
         (uav2LineEdit->hasAcceptableInput()) &&
         (uav2PortLineEdit->hasAcceptableInput())&&
                        (uav3LineEdit->hasAcceptableInput()) &&
         (uav3PortLineEdit->hasAcceptableInput())&&
         (uav12LineEdit->hasAcceptableInput()) &&
         (uav13LineEdit->hasAcceptableInput()) &&
                        (uav23LineEdit->hasAcceptableInput()) &&
         (altLineEdit->hasAcceptableInput()) &&(

        (StraightLineRadioButton->isOn())&&(
         (slNLineEdit->hasAcceptableInput()) &&
         (slELineEdit->hasAcceptableInput()) &&
         (slLLineEdit->hasAcceptableInput()) &&
         (slHLineEdit->hasAcceptableInput())   )
 ||
        (WaypointRadioButton->isOn())&&(
         (waypointFileLineEdit->hasAcceptableInput()) )
||
         (circleRadioButton->isOn())&&(
         (circleNLineEdit->hasAcceptableInput()) &&
         (circleELineEdit->hasAcceptableInput()) &&
         (circleRLineEdit->hasAcceptableInput())) ) );}
    else{
```

```
    connectButton->setEnabled( (uav1LineEdit->hasAcceptableInput()) &&
         (uav1PortLineEdit->hasAcceptableInput())&&
         (uav2LineEdit->hasAcceptableInput())//&&
       (orientationLineEdit->hasAcceptableInput())&&
           (uav2PortLineEdit->hasAcceptableInput())&&
         (uav12LineEdit->hasAcceptableInput())// &&
         (altLineEdit->hasAcceptableInput()) &&(

        (StraightLineRadioButton->isOn())&&(
         (slNLineEdit->hasAcceptableInput()) &&
         (slELineEdit->hasAcceptableInput()) &&
         (slLLineEdit->hasAcceptableInput()) &&
         (slHLineEdit->hasAcceptableInput())   )
 ||
        (WaypointRadioButton->isOn())&&(
         (waypointFileLineEdit->hasAcceptableInput()) )
||
         (circleRadioButton->isOn())&&(
         (circleNLineEdit->hasAcceptableInput()) &&
         (circleELineEdit->hasAcceptableInput()) &&
         (circleRLineEdit->hasAcceptableInput())) ) );}
}

void missionswindowImpl::connectButtonClicked()
{
     double x1 = 0.0;
     double x2 = 0.0;
     double x3 = 0.0;
     double y1 = 0.0;
     double y2 = 0.0;
     double y3 = 0.0;
     double z1 = 0.0;
     double z2 = 0.0;
     double z3 = 0.0;

     double line_north = 0.0;
     double line_east = 0.0;
     double heading = 0.0;
     double slope = 0.0;
     double length = 0.0;
     double orient = 0.0;
     QString waypoints_filename("waypoints.txt");
     double circle_north = 0.0;
     double circle_east = 0.0;;
     double radius = 0.0;
     double tol = 10.0;
     int connection_timeout = 20;

    numberOfUAV = uavSpinBox->value();
    //If there are 3 UAVs.
```

```
    if(numberOfUAV!=2){
  /*************************************************************************
   * NOTE: Currently software is configured for only 2UAVs.
   *       The code here is provided to make extension to 3UAVs easy.
   *************************************************************************/

uav1IP = uav1LineEdit->text();
uav2IP = uav2LineEdit->text();
uav3IP = uav3LineEdit->text();

uav1Port = uav1PortLineEdit->text().toInt();
uav2Port = uav2PortLineEdit->text().toInt();
uav3Port = uav3PortLineEdit->text().toInt();

delta_ij[1][2] = uav12LineEdit->text().toDouble();
delta_ij[1][3] = uav13LineEdit->text().toDouble();
delta_ij[2][3] = uav23LineEdit->text().toDouble();


setSafeBounds(delta_ij[1][2] - 0.2*delta_ij[1][2]);

if(StraightLineRadioButton->isOn()){
line_north = slNLineEdit->text().toDouble();
line_east   = slELineEdit->text().toDouble();
heading    = slHLineEdit->text().toDouble();
length   = slLLineEdit->text().toDouble();

}
if(WaypointRadioButton->isOn()){

waypoints_filename =  waypointFileLineEdit->text();


}
if(circleRadioButton->isOn()){
circle_north = circleNLineEdit->text().toDouble();
circle_east = circleELineEdit->text().toDouble();
radius = circleRLineEdit->text().toDouble();

}



heli1_x0.east  = inputs.X0;
  heli1_x0.north  = inputs.Y0 + (minseparation + 3.0);
heli1_x0.down = inputs.altitude;

heli2_x0.east  = inputs.X0 + (minseparation - 3.0);
heli2_x0.north  = inputs.Y0;
heli2_x0.down = inputs.altitude;
```

```
heli3_x0.east  = inputs.X0 - (minseparation + 3.0);
heli3_x0.north  = inputs.Y0;
heli3_x0.down = inputs.altitude;


// Set arguments for the processes

procT = new QProcess(this);
proc1 = new QProcess(this);
proc2 = new QProcess(this);
proc3 = new QProcess(this);

procT->addArgument("control/fly_target");
proc1->addArgument("control/fly_heli1");
proc2->addArgument("control/fly_heli2");
proc3->addArgument("control/fly_heli3");

    }

    //If there are 2UAV's
    else {

uav1IP = uav1LineEdit->text();
uav2IP = uav2LineEdit->text();

uav1Port = uav1PortLineEdit->text().toInt();
uav2Port = uav2PortLineEdit->text().toInt();

delta_ij[1][2] = uav12LineEdit->text().toDouble();

inputs.altitude = altLineEdit->text().toDouble();
inputs.formation = delta_ij[1][2];

orient = orientationLineEdit->text().toDouble();
        inputs.orientation = orient;
setSafeBounds(delta_ij[1][2] - 0.2*delta_ij[1][2]);

if(StraightLineRadioButton->isOn()){
    line_north = slNLineEdit->text().toDouble();
    line_east   = slELineEdit->text().toDouble();
    heading    = pi/180.0*(slHLineEdit->text().toDouble());
    length   = slLLineEdit->text().toDouble();

if( ((heading > 0.0 ) && (heading < pi/2.0 || heading == pi/2.0))||
((heading > -pi) && (heading < -pi/2.0 || heading == -pi/2.0)))
slope = tan(pi/2.0 - heading);
if( ((heading > pi/2.0 )&&(heading < pi))||
((heading > -pi/2.0) && (heading < 0.0)))
slope = -tan(pi - heading);
```

```
inputs.trajectory = LINE;
inputs.speed = speedSlider->value();
inputs.heading = heading;
inputs.slope = slope;
inputs.X0 = line_east;
inputs.Y0 = line_north;
  inputs.radius = 0.0;


heli1_x0.east  = inputs.X0;
  heli1_x0.north  = inputs.Y0 - (minseparation + 3.0);
heli1_x0.down = inputs.altitude;


heli2_x0.east  = inputs.X0;
  heli2_x0.north  = inputs.Y0 + (minseparation + 3.0);
heli2_x0.down = inputs.altitude;


}
if(WaypointRadioButton->isOn()){

waypoints_filename =  waypointFileLineEdit->text();

// Continuous waypoint generating algorithm goes here.


}
if(circleRadioButton->isOn()){
    circle_north = circleNLineEdit->text().toDouble();
    circle_east = circleELineEdit->text().toDouble();
    radius = circleRLineEdit->text().toDouble();
inputs.trajectory = CIRCLE;
inputs.speed = speedSlider->value();
inputs.radius = radius;
inputs.Y0 = circle_north;//0.0;
inputs.X0 = circle_east;//radius;


heli1_x0.east  = inputs.X0;
  heli1_x0.north  = inputs.Y0 - radius - (minseparation + 3.0);
heli1_x0.down = inputs.altitude;


heli2_x0.east  = inputs.X0;
  heli2_x0.north  = inputs.Y0 - radius + (minseparation + 3.0);
heli2_x0.down = inputs.altitude;


}
```

```
procT = new QProcess(this);
proc1 = new QProcess(this);
proc2 = new QProcess(this);

procT->addArgument("control/target_test");
proc1->addArgument("control/heli1_test");
proc2->addArgument("control/heli2_test");


}
     emit connected();

accept();

}


void missionswindowImpl::abortMission()
{
if(numberOfUAV == 3){
procT->tryTerminate();
QTimer::singleShot( 1000, procT, SLOT( kill() ) );
proc1->tryTerminate();
QTimer::singleShot( 1000, proc1, SLOT( kill() ) );
proc2->tryTerminate();
QTimer::singleShot( 1000, proc2, SLOT( kill() ) );
proc3->tryTerminate();
QTimer::singleShot( 1000, proc3, SLOT( kill() ) ); }

if(numberOfUAV ==2) {
procT->tryTerminate();
QTimer::singleShot( 1000, procT, SLOT( kill() ) );
proc1->tryTerminate();
QTimer::singleShot( 1000, proc1, SLOT( kill() ) );
proc2->tryTerminate();
QTimer::singleShot( 1000, proc2, SLOT( kill() ) );}

missionStatus = 0;

QMessageBox::information( 0, "Abort",
                          QString( "Mission aborted!" ) );

 }

void missionswindowImpl::haltAll()
{
shmdt(hp1);
shmdt(hp2);
shmdt(hp3);
shmdt(tp);
```

```
//Remove shared memory segments.
shmctl(shmh1_id,IPC_RMID,0);
  shmctl(shmh2_id,IPC_RMID,0);
shmctl(shmh3_id,IPC_RMID,0);
shmctl(shmt_id,IPC_RMID,0);

printf("Shared memory segments removed\n");
}

void missionswindowImpl::drawStraightLinePath()
{
   CurveData data;
   double X_intercept;
   double coord[4];

if( (inputs.heading > 0.0 )&&(inputs.heading < pi/2.0 ||
     inputs.heading == pi/2.0) ){
X_intercept = 1000.0;}
if( (inputs.heading > pi/2.0)&&(inputs.heading < pi) ){
X_intercept = 1000.0;}
if( (inputs.heading > -pi/2.0 || inputs.heading == -pi/2.0)&&
    (inputs.heading < 0.0) ){
X_intercept = -1000.0;}
if( (inputs.heading > -pi)&&(inputs.heading < -pi/2.0) ){
X_intercept = -1000.0;}

coord[0]  = inputs.X0;
coord[1]  = inputs.Y0;
coord[2]  = X_intercept;
coord[3]  = inputs.slope*(X_intercept-inputs.X0)+inputs.Y0;

if (inputs.heading == 0.0) {
coord[0]  = inputs.X0;
coord[1]  = inputs.Y0;
coord[2]  = inputs.X0;
coord[3]  = 1000.0;}

if ((inputs.heading == pi)||(inputs.heading == -pi)) {
coord[0]  = inputs.X0;
coord[1]  = inputs.Y0;
coord[2]  = inputs.X0;
coord[3]  = -1000.0;}

for (int i = 0; i < 4; ++i) {
data.push_back(coord[i]);
}
graph->drawWhat(LINE);
graph->setCurveData(0, data);
}

void missionswindowImpl::drawWaypointCurve()
```

```
{
    //Draw a smooth curve through waypoints.

}
void missionswindowImpl::drawCirclePath()
{
   CurveData data;
        double  coord[4] = {inputs.X0, inputs.Y0, 2.0*inputs.radius,
                            2.0*inputs.radius};

                for (int i = 0; i < 4; ++i) {
                    data.push_back(coord[i]);
                }
   graph->drawWhat(CIRCLE);
   graph->setCurveData(0, data);
}

void missionswindowImpl::plotPosition(double x, double y, int heli_no)
{
        static double last_point[2] = {0.0,0.0};  // initial position
double current_point[2];
double dx = 2.5;
double dy = 2.5;

current_point[0] = x;
current_point[1] = y;

CurveData data;
  double   coord[12] = {(x-dx), y, (x+dx), y, x, (y-dy),
                        x, (y+dy), x, y, 2.0*dx, 2.0*dy};//straightline
                for (int i = 0; i < 12; ++i) {
                    data.push_back(coord[i]);
                }

   graph->setPointData(heli_no, data);
last_point[0] = current_point[0];
        last_point[1] = current_point[1];

}

void missionswindowImpl::drawPath()
{
if(StraightLineRadioButton->isOn())
drawStraightLinePath();
if(circleRadioButton->isOn())
drawCirclePath();
if(WaypointRadioButton->isOn())
{
//plot waypoints
}
```

```
}
void missionswindowImpl::runAlgorithm()
{
bool uav1Started;
  bool uav2Started;
  bool uav3Started;
bool targetStarted;

// Start all processes

updateTimer->start(1000);
if (numberOfUAV == 3 ){

targetStarted  =  procT->start();
uav1Started = proc1->start();
uav2Started = proc2->start();
uav3Started = proc3->start();


if(!targetStarted||!uav1Started||!uav2Started||!uav3Started){
 QMessageBox::critical( 0, "Error",
                              QString("Could not start control programs! "
                                      "\nOne or more programs wont start"
      "\nRestart fcontrol and try again") );

//Terminated any started processes
procT->tryTerminate();
        QTimer::singleShot( 1000, procT, SLOT( kill() ) );
proc1->tryTerminate();
        QTimer::singleShot( 1000, proc1, SLOT( kill() ) );
proc2->tryTerminate();
        QTimer::singleShot( 1000, proc2, SLOT( kill() ) );
proc3->tryTerminate();
        QTimer::singleShot( 1000, proc3, SLOT( kill() ) );
 }
connect( proc3,SIGNAL(processExited()),this,SLOT(do_warning()));

}

else if (numberOfUAV == 2 ) {
initializeSharedMemory();
targetStarted  =  procT->start();
uav1Started = proc1->start();
uav2Started = proc2->start();


if(!targetStarted||!uav1Started||!uav2Started){
 QMessageBox::critical( 0, "Error",
                              QString("Could not start control programs! "
                                      "\nOne or more programs wont start"
```

```
      "\nRestart fcontrol and try again") );

//Terminated any started processes
procT->tryTerminate();
        QTimer::singleShot( 1000, procT, SLOT( kill() ) );
proc1->tryTerminate();
        QTimer::singleShot( 1000, proc1, SLOT( kill() ) );
proc2->tryTerminate();
        QTimer::singleShot( 1000, proc2, SLOT( kill() ) ); }
}

else {
 QMessageBox::critical( 0, "Error",
                              QString("Unknown number of UAV's!"
      "\nPlease select a valid number of UAV's") );
 }
 connect( proc1,SIGNAL(processExited()),this,SLOT(do_warning()));
 connect( proc2,SIGNAL(processExited()),this,SLOT(do_warning()));
 connect( procT,SIGNAL(processExited()),this,SLOT(do_warning()));
}


void missionswindowImpl::initializeSharedMemory()
{
    int world_shm_id1;
    int world_shm_id2;
    int world_shm_id3;
    int world_shm_id4;

    hp1  = initialize_heli1pos_shared_memory(HELI1_POS_SHM_KEY,&world_shm_id1);
    hp2 = initialize_heli2pos_shared_memory(HELI2_POS_SHM_KEY,&world_shm_id2);
    hp3 = initialize_heli3pos_shared_memory(HELI3_POS_SHM_KEY,&world_shm_id3);
    tp  = initialize_target_shared_memory(TARGET_POS_SHM_KEY,&world_shm_id4);

printf("x = %f, y = %f, z = %f.\n",hp1->position.x, hp1->position.y, hp1->position.z);
printf("x = %f, y = %f, z = %f.\n",hp2->position.x, hp2->position.y, hp2->position.z);
printf("x = %f, y = %f, z = %f.\n",hp3->position.x, hp3->position.y, hp3->position.z);
printf("x = %f, y = %f, z = %f.\n",tp->position.x, tp->position.y, tp->position.z);
}


Heli1_data * missionswindowImpl::initialize_heli1pos_shared_memory(int key,
                                                    int *shm_id_ret)
{
  int flag;
  Heli1_data *h1;

  printf("Attempt to get shared memory of heli 1 Position.\n");
  flag = IPC_CREAT | 0777;
  printf("%d",sizeof(Heli1_data));
  shmh1_id = shmget(key,sizeof(Heli1_data),flag);
```

```
  if(shmh1_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got shared memory of vehicle 1 Position\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of heli1.Position\n");
  h1 = (Heli1_data *)shmat(shmh1_id,0,0);

  if((int)h1 == -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli1 Position attached.\n");
  *shm_id_ret=shmh1_id;
  h1->position.x = heli1_x0.east;
h1->position.y = heli1_x0.north;
h1->position.z = 0.0;
h1->network.IP = uav1IP;
h1->network.Port = uav1Port;
h1->formation.data1 = delta_ij[1][2];
h1->formation.data2 = delta_ij[1][3];
h1->formation.data3 = delta_it[1];

  printf("Shared memory of h1 initialized.\n");
  //printf("x = %f, y = %f, z = %f.\n",h1->x, h1->y, h1->z);
  return(h1);

}

Heli2_data * missionswindowImpl::initialize_heli2pos_shared_memory(int key,
                                                    int *shm_id_ret)
{
  int flag;
  Heli2_data *h2;

  printf("Attempt to get Heli2 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh2_id = shmget(key,sizeof(Heli2_data),flag);
  if(shmh2_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli2 Shared memory 1\n");
  //attach shared memory
  printf("Attempt to attach to shared memory of Heli2.\n");
  h2 = (Heli2_data *)shmat(shmh2_id,0,0);
```

```
  if((int)h2== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of heli2 attached.\n");
  *shm_id_ret=shmh2_id;
  h2->position.x = heli2_x0.east;
h2->position.y = heli2_x0.north;
h2->position.z = 0.0;
h2->network.IP = uav2IP;
h2->network.Port = uav2Port;
h2->formation.data1 = delta_ij[2][1];
h2->formation.data2 = delta_ij[2][3];
h2->formation.data3 = delta_it[2];

  printf("Shared memory of h2 initialized.\n");
 // printf("x = %f, y = %f, z = %f.\n",h2->x, h2->y, h2->z);
 return(h2);

}

Heli3_data *missionswindowImpl::initialize_heli3pos_shared_memory(int key,
                                                                  int *shm_id_ret)
{
  int flag;
  Heli3_data *h3;

  printf("Attempt to get Heli3 shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmh3_id = shmget(key,sizeof(Heli3_data),flag);
  if(shmh3_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Heli3 Shared memory 1\n");
  //attach shared memory
  printf("Attempt to attach to shared memory of heli3.\n");
  h3 = (Heli3_data *)shmat(shmh3_id,0,0);
  if((int)h3== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of h3 attached.\n");
  *shm_id_ret=shmh3_id;

h3->position.x = heli3_x0.east;
h3->position.y = heli3_x0.north;
```

```
h3->position.z = 0.0;
h3->network.IP = uav3IP;
h3->network.Port = uav3Port;
h3->formation.data1 = delta_ij[3][1];
h3->formation.data2 = delta_ij[3][2];
h3->formation.data3 = delta_it[3];
  printf("Shared memory of h3 initialized.\n");
 // printf("x = %f, y = %f, z = %f.\n",h3->x, h3->y, h3->z);
 return(h3);

}

Target_data * missionswindowImpl::initialize_target_shared_memory(int key,
                                                                  int *shm_id_ret)
{
  int flag;
  Target_data *t1;

  printf("Attempt to get Target shared memory 1.\n");
  flag = IPC_CREAT | 0777;
  shmt_id = shmget(key,sizeof(Target_data),flag);
  if(shmt_id<0)
    {
      perror("error: shmget\n");
      exit(-1);
    }
  printf("Got Taget Shared memory 1\n");

  //attach shared memory
  printf("Attempt to attach to shared memory of vehicle2.\n");
  t1 = (Target_data *)shmat(shmt_id,0,0);

  if((int)t1== -1)
    {
      perror("error: shmat\n");
      exit(-1);
    }
  printf("Shared memory of target attached.\n");
  *shm_id_ret=shmt_id;
  t1->position.x = inputs.X0 + inputs.radius;
t1->position.y = inputs.Y0;
t1->position.z = 0.0;
t1->formation.trajectory = inputs.trajectory;
  t1->formation.altitude = inputs.altitude;
t1->formation.fmtn = inputs.formation;
t1->formation.circle_radius = inputs.radius;
t1->formation.target_speed = inputs.speed;
t1->formation.line_heading = inputs.heading;
  t1->formation.formation_orientation =   inputs.orientation;
t1->formation.line_slope = inputs.slope;
t1->formation.start_north = inputs.X0;
```

```
t1->formation.start_east = inputs.X0;;

    printf("Shared memory of target initialized.\n");
    //printf("x = %f, y = %f, z = %f.\n",t1->x, t1->y, t1->z);
 return(t1);

}
/*********************************************************************/
/*****************************************plotter.h*****************/
* This file implements the plotter object.
* This code was partly taken from the QT3 documentation.
*********************************************************************/
#ifndef PLOTTER_H
#define PLOTTER_H

#include <qpixmap.h>
#include <qwidget.h>

#include <map>
#include <vector>

class QToolButton;
class PlotSettings;

enum mission {LINE=1, BOX, CIRCLE, NOTHING};

typedef std::vector<double> CurveData;

class Plotter : public QWidget
{
    Q_OBJECT
public:
    Plotter(QWidget *parent = 0, const char *name = 0,
            WFlags flags = 0);

    void setPlotSettings(const PlotSettings &settings);
    void setCurveData(int id, const CurveData &data);
    void setPointData(int id, const CurveData &data);
    void setTrailData(int id, const CurveData &data);
    void drawWhat(mission);
    void clearCurve(int id);
    QSize minimumSizeHint() const;
    QSize sizeHint() const;

public slots:
    void zoomIn();
    void zoomOut();

protected:
    void paintEvent(QPaintEvent *event);
    void resizeEvent(QResizeEvent *event);
```

```
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent *event);
    void mouseReleaseEvent(QMouseEvent *event);
    void keyPressEvent(QKeyEvent *event);
    void wheelEvent(QWheelEvent *event);

private:
    void updateRubberBandRegion();
    void refreshPixmap();
    void drawGrid(QPainter *painter);
    void drawLineMission(QPainter *painter);
    void drawCircleMission(QPainter *painter);
    void drawBoxMission(QPainter *painter);
    void drawPositions(QPainter *painter);
    void drawCurves(QPainter *painter);

    enum { Margin = 40 };
    mission trajectory;
    //int k;

    QToolButton *zoomInButton;
    QToolButton *zoomOutButton;
    std::map<int, CurveData> curveMap;
    std::map<int, CurveData> pointMap;
    std::map<int, CurveData> trailMap;
    std::vector<PlotSettings> zoomStack;
    int curZoom;
    bool rubberBandIsShown;
    QRect rubberBandRect;
    QPixmap pixmap;
};

class PlotSettings
{
public:
    PlotSettings();

    void scroll(int dx, int dy);
    void adjust();
    double spanX() const { return maxX - minX; }
    double spanY() const { return maxY - minY; }

    double minX;
    double maxX;
    int numXTicks;
    double minY;
    double maxY;
    int numYTicks;


private:
```

```
    void adjustAxis(double &min, double &max, int &numTicks);
};

#endif
/*************************************************************/
/*********************************plotter.cpp*****************/
#include <qpainter.h>
#include <qstyle.h>
#include <qtoolbutton.h>

#include <cmath>
using namespace std;

#include "plotter.h"

Plotter::Plotter(QWidget *parent, const char *name, WFlags flags)
    : QWidget(parent, name, flags | WNoAutoErase)
{
    setPaletteBackgroundColor(black);
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    setFocusPolicy(StrongFocus);
    rubberBandIsShown = false;

    zoomInButton = new QToolButton(this);
    zoomInButton->setIconSet(QPixmap::fromMimeSource("zoomin.png"));
    zoomInButton->adjustSize();
    connect(zoomInButton, SIGNAL(clicked()), this, SLOT(zoomIn()));

    zoomOutButton = new QToolButton(this);
    zoomOutButton->setIconSet(
            QPixmap::fromMimeSource("zoomout.png"));
    zoomOutButton->adjustSize();
    connect(zoomOutButton, SIGNAL(clicked()), this, SLOT(zoomOut()));

    drawWhat(NOTHING);
    setPlotSettings(PlotSettings());

}

void Plotter::drawWhat(mission track)
{
 trajectory = track;
}
void Plotter::setPlotSettings(const PlotSettings &settings)
{
    zoomStack.resize(1);
    zoomStack[0] = settings;
    curZoom = 0;
    zoomInButton->hide();
    zoomOutButton->hide();
    refreshPixmap();
```

```
}

void Plotter::zoomOut()
{
    if (curZoom > 0) {
        --curZoom;
        zoomOutButton->setEnabled(curZoom > 0);
        zoomInButton->setEnabled(true);
        zoomInButton->show();
        refreshPixmap();
    }
}

void Plotter::zoomIn()
{
    if (curZoom < (int)zoomStack.size() - 1) {
        ++curZoom;
        zoomInButton->setEnabled(
                curZoom < (int)zoomStack.size() - 1);
        zoomOutButton->setEnabled(true);
        zoomOutButton->show();
        refreshPixmap();
    }
}

void Plotter::setCurveData(int id, const CurveData &data)
{
    curveMap[id] = data;
    refreshPixmap();
}

void Plotter::setPointData(int id, const CurveData &data)
{
    pointMap[id] = data;
    refreshPixmap();
}

void Plotter::setTrailData(int id, const CurveData &data)
{
    trailMap[id] = data;
    refreshPixmap();
}

void Plotter::clearCurve(int id)
{
    curveMap.erase(id);
    refreshPixmap();
}

QSize Plotter::minimumSizeHint() const
{
```

```
    return QSize(4 * Margin, 4 * Margin);
}

QSize Plotter::sizeHint() const
{
    return QSize(8 * Margin, 6 * Margin);
}

void Plotter::paintEvent(QPaintEvent *event)
{
    QMemArray<QRect> rects = event->region().rects();
    for (int i = 0; i < (int)rects.size(); ++i)
        bitBlt(this, rects[i].topLeft(), &pixmap, rects[i]);

    QPainter painter(this);

    if (rubberBandIsShown) {
        painter.setPen(colorGroup().dark());
        painter.drawRect(rubberBandRect.normalize());
    }
    if (hasFocus()) {
        style().drawPrimitive(QStyle::PE_FocusRect, &painter,
                              rect(), colorGroup(),
                              QStyle::Style_FocusAtBorder,
                              colorGroup().dark());
    }
}

void Plotter::resizeEvent(QResizeEvent *)
{
    int x = width() - (zoomInButton->width()
                       + zoomOutButton->width() + 10);
    zoomInButton->move(x, 5);
    zoomOutButton->move(x + zoomInButton->width() + 5, 5);
    refreshPixmap();
}

void Plotter::mousePressEvent(QMouseEvent *event)
{
    if (event->button() == LeftButton) {
        rubberBandIsShown = true;
        rubberBandRect.setTopLeft(event->pos());
        rubberBandRect.setBottomRight(event->pos());
        updateRubberBandRegion();
        setCursor(crossCursor);
    }
}

void Plotter::mouseMoveEvent(QMouseEvent *event)
{
    if (event->state() & LeftButton) {
```

```
        updateRubberBandRegion();
        rubberBandRect.setBottomRight(event->pos());
        updateRubberBandRegion();
    }
}

void Plotter::mouseReleaseEvent(QMouseEvent *event)
{
    if (event->button() == LeftButton) {
        rubberBandIsShown = false;
        updateRubberBandRegion();
        unsetCursor();

        QRect rect = rubberBandRect.normalize();
        if (rect.width() < 4 || rect.height() < 4)
            return;
        rect.moveBy(-Margin, -Margin);

        PlotSettings prevSettings = zoomStack[curZoom];
        PlotSettings settings;
        double dx = prevSettings.spanX() / (width() - 2 * Margin);
        double dy = prevSettings.spanY() / (height() - 2 * Margin);
        settings.minX = prevSettings.minX + dx * rect.left();
        settings.maxX = prevSettings.minX + dx * rect.right();
        settings.minY = prevSettings.maxY - dy * rect.bottom();
        settings.maxY = prevSettings.maxY - dy * rect.top();
        settings.adjust();

        zoomStack.resize(curZoom + 1);
        zoomStack.push_back(settings);
        zoomIn();
    }
}

void Plotter::keyPressEvent(QKeyEvent *event)
{
    switch (event->key()) {
    case Key_Plus:
        zoomIn();
        break;
    case Key_Minus:
        zoomOut();
        break;
    case Key_Left:
        zoomStack[curZoom].scroll(-1, 0);
        refreshPixmap();
        break;
    case Key_Right:
        zoomStack[curZoom].scroll(+1, 0);
        refreshPixmap();
        break;
```

```
    case Key_Down:
        zoomStack[curZoom].scroll(0, -1);
        refreshPixmap();
        break;
    case Key_Up:
        zoomStack[curZoom].scroll(0, +1);
        refreshPixmap();
        break;
    default:
        QWidget::keyPressEvent(event);
    }
}

void Plotter::wheelEvent(QWheelEvent *event)
{
    int numDegrees = event->delta() / 8;
    int numTicks = numDegrees / 15;

    if (event->orientation() == Horizontal)
        zoomStack[curZoom].scroll(numTicks, 0);
    else
        zoomStack[curZoom].scroll(0, numTicks);
    refreshPixmap();
}

void Plotter::updateRubberBandRegion()
{
    QRect rect = rubberBandRect.normalize();
    update(rect.left(), rect.top(), rect.width(), 1);
    update(rect.left(), rect.top(), 1, rect.height());
    update(rect.left(), rect.bottom(), rect.width(), 1);
    update(rect.right(), rect.top(), 1, rect.height());
}

void Plotter::refreshPixmap()
{
    pixmap.resize(size());
    pixmap.fill(this, 0, 0);
    QPainter painter(&pixmap, this);

    drawGrid(&painter);
    drawPositions(&painter);
    drawCurves(&painter);
    if(trajectory == LINE)
     drawLineMission(&painter);
    if(trajectory == BOX)
        drawBoxMission(&painter);
    if(trajectory == CIRCLE)
drawCircleMission(&painter);
    if(trajectory == NOTHING)
{;}
```

```
    update();

}

void Plotter::drawGrid(QPainter *painter)
{

    setBackgroundMode(PaletteLight);
    QRect rect(Margin, Margin,
               width() - 2 * Margin, height() - 2 * Margin);
    PlotSettings settings = zoomStack[curZoom];
    QPen quiteDark = colorGroup().dark().dark();
    QPen light = colorGroup().dark().dark();
    QString easting = QString("Easting");
    QString northing = QString("Northing");

    for (int i = 0; i <= settings.numXTicks; ++i) {
        int x = rect.left() + (i * (rect.width() - 1)
                               / settings.numXTicks);
        double label = settings.minX + (i * settings.spanX()
                                        / settings.numXTicks);
        painter->setPen(quiteDark);
        painter->drawLine(x, rect.top(), x, rect.bottom());
        painter->setPen(light);
        painter->drawLine(x, rect.bottom(), x, rect.bottom() + 5);
        painter->drawText(x - 50, rect.bottom() + 5, 100, 15,
                          AlignHCenter | AlignTop,
                          QString::number(label));


    }
    for (int j = 0; j <= settings.numYTicks; ++j) {
        int y = rect.bottom() - (j * (rect.height() - 1)
                                 / settings.numYTicks);
        double label = settings.minY + (j * settings.spanY()
                                        / settings.numYTicks);
        painter->setPen(quiteDark);
        painter->drawLine(rect.left(), y, rect.right(), y);
        painter->setPen(light);
        painter->drawLine(rect.left() - 5, y, rect.left(), y);
        painter->drawText(rect.left() - Margin, y - 10,
                          Margin - 5, 20,
                          AlignRight | AlignVCenter,
                          QString::number(label));

    }

    painter->drawText(300, rect.bottom() + 15, 100, 15,
                          AlignHCenter | AlignTop,easting);
    painter->drawText(rect.left() - Margin,  10,
```

```
                        Margin+10 , 20,
                        AlignHCenter ,
                        northing);

    painter->drawRect(rect);
}

void Plotter::drawLineMission(QPainter *painter)
{

    QPen thickPen(darkGreen,2);
    painter->setPen(thickPen);

    PlotSettings settings = zoomStack[curZoom];
    QRect rect(Margin, Margin,
               width() - 2 * Margin, height() - 2 * Margin);

    painter->setClipRect(rect.x() + 1, rect.y() + 1,
                         rect.width() - 2, rect.height() - 2);

    map<int, CurveData>::const_iterator it = curveMap.begin();
    while (it != curveMap.end()) {
        int id = (*it).first;
        const CurveData &data = (*it).second;
        int numPoints = 0;
        int maxPoints = data.size() / 2;
        QPointArray points(maxPoints);

        for (int i = 0; i < maxPoints; ++i) {
            double dx = data[2 * i] - settings.minX;
            double dy = data[2 * i + 1] - settings.minY;
            double x = rect.left() + (dx * (rect.width() - 1)
                                      / settings.spanX());
            double y = rect.bottom() - (dy * (rect.height() - 1)
                                        / settings.spanY());
            if (fabs(x) < 32768 && fabs(y) < 32768) {
                points[numPoints] = QPoint((int)x, (int)y);
                ++numPoints;
            }
        }
        points.truncate(numPoints);

        painter->drawLine(points[0],points[1]);
        ++it;
    }

}
void Plotter::drawCircleMission(QPainter *painter)
{
    QPen thickPen(darkGreen,2);
    painter->setPen(thickPen);
```

```
    PlotSettings settings = zoomStack[curZoom];
    QRect rect(Margin, Margin,
               width() - 2 * Margin, height() - 2 * Margin);

    painter->setClipRect(rect.x() + 1, rect.y() + 1,
                         rect.width() - 2, rect.height() - 2);

    map<int, CurveData>::const_iterator it = curveMap.begin();
    while (it != curveMap.end()) {
        int id = (*it).first;
        const CurveData &data = (*it).second;
        int numPoints = 0;
        int maxPoints = data.size() / 2;
        QPointArray points(maxPoints);

        for (int i = 0; i < maxPoints; ++i) {
            double dx = data[2 * i] - settings.minX*(1-i) - data[2]*(1-i)/2.0;
            double dy = data[2 * i + 1] - settings.minY*(1-i) + data[3]*(1-i)/2.0;

            double x = (1-i)*rect.left() + (dx * (rect.width() - 1)
                                        / settings.spanX());
            double y = (1-i)*rect.bottom() + (-1+2*i)*(dy * (rect.height() - 1)
                                          / settings.spanY());
            if (fabs(x) < 32768 && fabs(y) < 32768) {
                points[numPoints] = QPoint((int)x, (int)y);
                ++numPoints;
            }
        }
        points.truncate(numPoints);

        painter->drawEllipse(QRect(points[0].x(),points[0].y(),
        points[1].x(),points[1].y()));
        ++it;
    }
}
void Plotter::drawBoxMission(QPainter *painter)
{
    QPen thickPen(darkGreen,2);
    painter->setPen(thickPen);

    PlotSettings settings = zoomStack[curZoom];
    QRect rect(Margin, Margin,
               width() - 2 * Margin, height() - 2 * Margin);

    painter->setClipRect(rect.x() + 1, rect.y() + 1,
                         rect.width() - 2, rect.height() - 2);

    map<int, CurveData>::const_iterator it = curveMap.begin();
    while (it != curveMap.end()) {
        int id = (*it).first;
```

```
        const CurveData &data = (*it).second;
        int numPoints = 0;
        int maxPoints = data.size() / 2;
        QPointArray points(maxPoints);

      for (int i = 0; i < maxPoints; ++i) {
          double dx = data[2 * i] - settings.minX*(1-i) - data[2]*(1-i)/2.0;
          double dy = data[2 * i + 1] - settings.minY*(1-i) + data[3]*(1-i)/2.0;

          double x = (1-i)*rect.left() + (dx * (rect.width() - 1)
                                        / settings.spanX());
          double y = (1-i)*rect.bottom() + (-1+2*i)*(dy * (rect.height() - 1)
                                         / settings.spanY());
          if (fabs(x) < 32768 && fabs(y) < 32768) {
              points[numPoints] = QPoint((int)x, (int)y);
              ++numPoints;
            }
        }
        points.truncate(numPoints);

        painter->drawRoundRect(points[0].x(),points[0].y(),
        points[1].x(),points[1].y());
        ++it;
    }
}
void Plotter::drawPositions(QPainter *painter)
{
 QPen thickPen(red);
 QPen thickBlackPen(black);
         QPen textPen(black);

         thickPen.setWidth(4);
         thickPen.setCapStyle(FlatCap);

 thickBlackPen.setWidth(4);
         thickBlackPen.setCapStyle(FlatCap);

    painter->setPen(thickPen);

    PlotSettings settings = zoomStack[curZoom];
    QRect rect(Margin, Margin,
               width() - 2 * Margin, height() - 2 * Margin);

    painter->setClipRect(rect.x() + 1, rect.y() + 1,
                         rect.width() - 2, rect.height() - 2);

    map<int, CurveData>::const_iterator it = pointMap.begin();
    while (it != pointMap.end()) {
        int id = (*it).first;
        const CurveData &data = (*it).second;
        int numPoints = 0;
```

```
        int maxPoints = data.size() / 2;
        double dx;
        double dy;
double x;
double y;
        QPointArray points(maxPoints);


        for (int i = 0; i < maxPoints; ++i) {

if ( i<4 ){
          dx = data[2 * i] - settings.minX;
          dy = data[2 * i + 1] - settings.minY;
          x = rect.left() + (dx * (rect.width() - 1)
                                    / settings.spanX());
          y = rect.bottom() - (dy * (rect.height() - 1)
                                     / settings.spanY());}

else{
    dx = data[2 * i] - settings.minX*(5-i) - data[10]*(5-i)/2.0;
          dy = data[2 * i + 1] - settings.minY*(5-i) + data[11]*(5-i)/2.0;

          x = (5-i)*rect.left() + (dx * (rect.width() - 1)
                                    / settings.spanX());
          y = (5-i)*rect.bottom() + (-9+2*i)*(dy * (rect.height() - 1)
                                     / settings.spanY());}

          if (fabs(x) < 32768 && fabs(y) < 32768) {
              points[numPoints] = QPoint((int)x, (int)y);
              ++numPoints;
            }
        }
        points.truncate(numPoints);

if ((int)id == 5)painter->setPen(thickBlackPen);

        painter->drawLine(points[0],points[1]);
painter->drawLine(points[2],points[3]);
painter->drawEllipse(QRect(points[4].x(),points[4].y(),
points[5].x(),points[5].y()));

painter->setPen(textPen);
if ((int)id == 0) painter->drawText((points[4].x() + 5.0),(points[4].y()),"1");
if ((int)id == 1) painter->drawText((points[4].x() + 5.0),(points[4].y()),"2");

painter->setPen(thickPen);
       painter->drawEllipse(QRect(points[0].x(),points[0].y(),
       points[1].x(),points[1].y()));
        ++it;
    }
}
```

```
void Plotter::drawCurves(QPainter *painter)
{
    static const QColor colorForIds[6] = {
        magenta, blue, cyan, red, green, yellow
    };

    QPen lightPen;
    PlotSettings settings = zoomStack[curZoom];
    QRect rect(Margin, Margin,
               width() - 2 * Margin, height() - 2 * Margin);

    painter->setClipRect(rect.x() + 1, rect.y() + 1,
                         rect.width() - 2, rect.height() - 2);

    map<int, CurveData>::const_iterator it = trailMap.begin();
    while (it != trailMap.end()) {
        int id = (*it).first;
        const CurveData &data = (*it).second;
        int numPoints = 0;
        int maxPoints = data.size() / 2;
        QPointArray points(maxPoints);


        for (int i = 0; i < maxPoints; ++i) {
            double dx = data[2 * i] - settings.minX;
            double dy = data[2 * i + 1] - settings.minY;
            double x = rect.left() + (dx * (rect.width() - 1)
                                      / settings.spanX());
            double y = rect.bottom() - (dy * (rect.height() - 1)
                                        / settings.spanY());
            if (fabs(x) < 32768 && fabs(y) < 32768) {
                points[numPoints] = QPoint((int)x, (int)y);
                ++numPoints;
            }
        }
        points.truncate(numPoints);
      lightPen.setColor(colorForIds[(uint)id % 6]);
lightPen.setWidth(2);
        painter->setPen(lightPen);
painter->drawPolyline(points);

        ++it;
    }

}

PlotSettings::PlotSettings()
{
    minX = 0.0;
    maxX = 100.0;
```

```
    numXTicks = 10;

    minY = 0.0;
    maxY = 100.0;
    numYTicks = 10;
}

void PlotSettings::scroll(int dx, int dy)
{
    double stepX = spanX() / numXTicks;
    minX += dx * stepX;
    maxX += dx * stepX;

    double stepY = spanY() / numYTicks;
    minY += dy * stepY;
    maxY += dy * stepY;
}

void PlotSettings::adjust()
{
    adjustAxis(minX, maxX, numXTicks);
    adjustAxis(minY, maxY, numYTicks);
}

void PlotSettings::adjustAxis(double &min, double &max,
                              int &numTicks)
{
    const int MinTicks = 4;
    double grossStep = (max - min) / MinTicks;
    double step = pow(10, floor(log10(grossStep)));

    if (5 * step < grossStep)
        step *= 5;
    else if (2 * step < grossStep)
        step *= 2;

    numTicks = (int)(ceil(max / step) - floor(min / step));
    min = floor(min / step) * step;
    max = ceil(max / step) * step;
}
```

BIBLIOGRAPHY

[1] J. Yao, R. Ordóñez, and V. Gazi, "Swarm tracking using artificial potentials and sliding mode control," in *Submitted to the Conference Decision and Control.*, 2006.

[2] E. Fiorelli, P. Bhatta, N. E. Leonard, and I. Shulman, "Adaptive sampling using feedback control of an autonomous underwater glider fleet," in *Proc. 13th Int. Symp. on Unmanned Untethered Submersible Technology*, August 2003.

[3] O. Khatib and J. L. Maitre, "Dynamic control of manipulator operating in complex environment," *Proceedings of third international CISM-ITOMM Symposium*, pp. 267–282, Sept. 1978.

[4] H. Yamaguchi, "A cooperative hunting behavior by mobile-robot troops," *The International Journal of Robotics Research*, vol. 18, pp. 931–940, September 1999.

[5] N. E. Leonard and E. Fiorelli, "Virtual leaders, artificial potentials and coordinated control of groups," in *Proc. 40th IEEE Conf. Decision and Control*, (Orlando, FL), pp. 2968–2973, December 2001.

[6] V. Gazi, "Swarm aggregations using artificial potentials and sliding mode control," *IEEE Transactions on Robotics*, vol. 21, pp. 1208–1214, December 2005.

[7] V. Gazi and R. Ordóñez, "Target tracking using artificial potentials and sliding mode control," in *Proc. American Control Conference*, (Boston, MA), pp. 5588–5593, June-July 2004.

[8] V. Gazi and K. M. Passino, "Stability analysis of social foraging swarms," *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, vol. 34, pp. 539–557, February 2004.

[9] D. H. Kim, H. O. Wang, G. Ye, and S. Shin, "Decentralized control of autonomous swarm systems using artificial potential functions: Analytical design guidelines," in *IEEE Conference on Decision and Control*, December 2004.

[10] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transaction on Robotics and Automation*, vol. 8, pp. 501–518, October 1992.

[11] L. Pimenta, A. Fonseca, G. Pereira, and R. Mesquita, "Robot navigation based on electrostatic field computation," *IEEE Transaction on Magnetics*, vol. 42, April 2006.

[12] J. P. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *Proc. 1998 IEEE Int. Conf. Robotics and Automation*, (Leuven, Belgium), pp. 2864–2869, May 1998.

[13] J. P. Desai, J. Ostrowski, and V. Kumar, "Modeling and control of formations of nonholonomic mobile robots," in *IEEE Trans. on. Robotics and Automation*, vol. 17, pp. 905–908, December 2001.

[14] R. Olfati-Saber and R. M. Murray, "Distributed cooperative control of multiple vehicle formations using structural potential functions," in *Proc. of the IFAC World Congress*, (Barcelona, Spain), June 2002.

[15] J. Guldner and V. Utkin, "Sliding mode control for an obstacle avoidance strategy based on an harmonic potential field," in *Proc. Of Conf. Decision Contr.*, (San Antonio, Texa), pp. 424–429, December 1993.

[16] J. Guldner and V. Utkin, "Sliding mode control for gradient tracking and robot navigation using artificial potential fields," *IEEE Trans. On Robotics and Automation*, vol. 11, no. 2, pp. 247–254, 1995.

[17] V. Gazi and K. M. Passino, "A class of attraction/repulsion functions for stable swarm aggregations," *International Journal of Control*, vol. 77, pp. 1567–1579, December 2004.

[18] V. Gazi and K. M. Passino, "Stability analysis of swarms," *IEEE Transactions on Automatic Control*, vol. 48, pp. 692–697, April 2003.